⑲日本国特許庁(JP)

⑫公表特許公報(A)　平4－503582

⑬公表　平成4年(1992)6月25日

⑤Int. Cl.⁵	識別記号	庁内整理番号	審査請求　未請求
G 06 F 9/38	370 A	8725－5B	予備審査請求　有　部門(区分)　6(3)
15/16	370 Z	9190－5L	

(全 30 頁)

⑭発明の名称　コンピュータの分散型パイプライン制御装置及び方法

⑳特　願　平2－504389
㉒㉓出　願　平2(1990)2月21日
㉕翻訳文提出日　平3(1991)8月23日
㉖国際出願　PCT/US90/00938
㉗国際公開番号　WO90/10267
㉘国際公開日　平2(1990)9月7日

優先権主張　㉜1989年2月24日㉝米国(US)㉛315,358

⑦発明者　マクファーランド　ハロルド　エル　アメリカ合衆国　カリフオルニア　95129　サン　ホセ　アツプルウツド　ドライブ　4750

⑦出願人　ネクスジエン　マイクロシステムズ　アメリカ合衆国　カリフオルニア　95131　サン　ホセ　ノースフアースト　ストリート　2202

⑦代理人　弁理士　鈴木　弘男

⑧指定国　AT(広域特許),BE(広域特許),CH(広域特許),DE(広域特許),DK(広域特許),ES(広域特許),FR(広域特許),GB(広域特許),IT(広域特許),JP,KR,LU(広域特許),NL(広域特許),SE(広域特許)

最終頁に続く

〔請求の範囲〕

1．発行されるとそれぞれ1つの未済演算の状態を達成する一連の演算を発行する装置と、

未済演算の少なくとも若干をそれぞれが実行できる複数の機能ユニットと、

割り当てられたタグを検査することによって2つの未済演算の相対年齢を決定できるように順序を付けたタグの集合の一員であるタグを各未済演算に割り当てる装置と、

所与の未済演算が完了する時点を決定する装置と、

未済タグの独特さを保証するように未済演算の数を制限する装置と

を具備することを特徴とするコンピュータプロセッサ。

2．上記制限する装置が、一時に多くともn演算を未済可能とすることを許容し、

上記タグが2nより大きいか等しいある範囲に亘って順次に発行され、

2つの未済演算の相対年齢をそれらのタグの符号付き比較によって決定できるようにした

請求項1に記載のコンピュータプロセッサ。

3．一時に多くともn演算が未済であり、

タグを、多くとも1つのビットがセットされているNビットベクトル（ここにNはnに等しいか大きい）として表し、

未済演算の集合をNビットベクトル内のビットの関連群によって表すようにタグを発行する

請求項1に記載のコンピュータプロセッサ。

4．異常状態と、未済演算の集合を流出させることの決定とに応答し、未済演算の集合を流出させることを表すNビットベクトルを前記機能ユニットへ通信する装置をも具備する請求項3に記載のコンピュータプロセッサ。

5．集合を流出させることを表す上記Nビットベクトルが、流出させる演算の集合に対応するビット集合を有する請求項4に記載のコンピュータプロセッサ。

6．少なくとも最古の未済演算の正常完了と、未済演算の集合を引退させることの決定とに応答し、未済演算の集合を引退させることを表すNビットベクトルを前記機能ユニットへ通信する装置をも具備する請求項3に記載のコンピュータプロセッサ。

7．集合を引退させることを表す上記Nビットベクトルが、引退させる演算の集合に対応するビット集合を有する請求項6に記載のコンピュータプロセッサ。

8．上記機能ユニットが複数の半導体チップ内に実現されている請求項1に記載のコンピュータプロセッサ。

9．異常状態に応答して所与のタグより遅く発行されたタグを有する全ての未済演算を流出させる装置をも具備する請求項1に記載のコンピュータプロセッサ。

10．群内の何れかの演算の流出がその群内の全ての演算の流出をもたらすように隣接する演算をグループ化する装置をも具備する請求項9に記載のコンピュータプロセッサ。

11．上記演算の少なくとも若干がが分岐演算であり、
未済分岐演算の結果を予測する装置と、
未済分岐演算の誤った予測を検出する装置と、
誤って予測された分岐演算の結果として発行された全ての未済演算を流出させる装置
をも具備する請求項1に記載のコンピュータプロセッサ。

12．最古の未済演算を決定する装置と、
未済演算と引退させる演算との間の境界をマークするタグを準備することによって、演算の引退が成功したことを上記機能ユニットの少なくとも若干へ通知する装置
をも具備する請求項1に記載のコンピュータプロセッサ。

13．群内の全ての演算の引退が可能になった時にのみその群内の何れかの演算の引退が進行されるように隣接演算をグループ化する装置
をも具備する請求項12に記載のコンピュータプロセッサ。

14．少なくともそれらの源演算が未済ではあるが引退はしていない期間の間メモリ書き込みを緩衝する装置と、
それらの源演算が流出した時に、緩衝されている書き込みを流出させる装置と、
それらの源演算が引退した時に、緩衝されている書き込みのキャッシュまたはメモリへの配置を完了させる装置
をも具備する請求項12に記載のコンピュータプロセッサ。

15．書き込みデータが流出されるか、またはキャッシュもしくはメモリ内に配置される前に、緩衝されている書き込みデータを同値の読み出し演算へ戻す装置
をも具備する請求項14に記載のコンピュータプロセッサ。

16．上記発行装置がm個のプログラマ可視のレジスタを含む命令集合アーキテクチャを有する入力命令に応答し、少なくとも若干の演算がこれらのレジスタの1つを変更し、
未済のレジスタ変更演算の数をnに制限する装置と、
少なくとも（m＋n）個の物理レジスタと、
プログラマ可視のレジスタを物理レジスタへ写像する装置
をも具備する請求項1に記載のコンピュータプロセッサ。

17．物理レジスタを変更した演算が成功裏に引退してしまうまで物理レジスタが再使用されないことを保証する装置と、
仮想から物理への写像を異常状態が検出された時の先行状態に復元し、従ってプログラマ可視のレジスタの内容を復元する装置
をも具備する請求項16に記載のコンピュータプロセッサ。

18．レジスタ変更は、プログラマが指定した順序以外で発生することが許される
請求項17に記載のコンピュータプロセッサ。

19．各物理レジスタに対応付けられた有効ビットと、
演算への入力として要求される全ての物理レジスタの有効ビットを調べる装置と、
有効ビットがクリアされている少なくとも1つの物理レジスタの演算の実行を遅延させる装置
をも具備する請求項17に記載のコンピュータプロセッサ。

20．インタロックが要求された時、直接的に影響された特定の機能ユニットだけがインタロックされた演算の実行を遅延させる
請求項1に記載のコンピュータプロセッサ。

21．若干のまたは全ての機能ユニットは、若干のまたは全ての演算が完了すると終了を通知し、
これらの終了は、完了した演算のタグを決定するのに充分な情報を提供し、
これらの終了は、演算の処理中に機能ユニットによって検出された最高優先順位の異常状態（もしあれば）を決定するのに充分な情報を提供する
請求項1に記載のコンピュータプロセッサ。

22．各未済演算毎に機能ユニットによって通知される全ての終了に優先順位を付ける装置と、
最古の異常に終了した演算を選択する装置と、
最古の異常に終了した演算の最高優先順位終了に対する正しい応答を決定する装置、
をも具備する請求項21に記載のコンピュータプロセッサ。

23．機能ユニットからの終了を、演算が発行された順序とは異なる順序で通知できる
請求項21に記載のコンピュータプロセッサ。

24．特定の型の書き込みを処理するためにそれぞれ割り当てられる複数の書き込みバッファ待ち行列と、
各書き込み待ち行列毎に最高優先順位を選択する装置と、
最高優先順位待ち行列を選択する装置と
を設けることによってインタロック回避及び性能を向上せしめたことを特徴とするコンピュータプロセッサ。

25．上記書き込みが順不同で発生することを許される請求項24に記載のコンピュータプロセッサ。

26．命令を含む入力流に応答してこの入力流内の命令を一

道の演算に変換する装置と、

これらの演算の少なくとも若干をそれぞれが実行できる複数の機能ユニット、

これらの演算を機能ユニットの少なくとも若干に通信する（このように通信される演算を未済演算と言う）装置と、

タグを各未済演算に順次に割り当てる装置と、

機能ユニットによる演算の終了に関する情報を各未済演算毎に維持する装置と、

各機能ユニットに組合わされ、通信されて来た各演算についてそれが何時終了したかを決定し、この終了情報をその演算のタグと共に前記維持装置に通信する装置と、

最古の未済演算を決定する装置と、

最古の未済演算の表示を機能ユニットに通信する装置と、

少なくとも最古の未済演算に関する終了情報に応答し、その演算の終了情報が全ての機能ユニットがその演算を正常に終了させたことを示している場合に限ってその演算の引退を許容する装置と、

少なくとも最古の未済演算の引退に応答し、このように引退した演算が最早未済ではないことを表すように最古の未済演算の表示を更新する装置と

を具備することを特徴とするコンピュータプロセッサ。

２７．所与の未済演算が異常に終了したことの情報に応答して、流出させるべき演算の群を指定する打ち切りタグを機能ユニットへ通信する装置と、

各機能ユニットに組合わされ、打ち切りタグによって指定された全ての未済演算を流出させる装置と、

打ち切りタグによって指定された演算から未済演算の指定を削除する装置と、

流出した最古の演算のタグに等しい値から始まるさらなるタグの割り当てを前記タグ割り当て装置に開始させる装置

をも具備する請求項２６に記載のコンピュータプロセッサ。

２８．打ち切りタグが、流出させられた最古の演算のタグに等しい請求項２７に記載のコンピュータプロセッサ。

２９．命令を含む入力流に応答してこの入力流内の命令を一連の演算に変換する装置と、

これらの演算の少なくとも若干をそれぞれが実行できる複数の機能ユニット、

これらの演算を機能ユニットの少なくとも若干に通信する（このように通信される演算を未済演算と言う）装置と、

未済演算の数を所定の最大数に制限する装置と、

タグを各未済演算に順次に割り当てる装置と、

機能ユニットによる演算の終了に関する情報を各未済演算毎に維持する装置と、

各機能ユニットに組合わされ、通信されて来た各演算についてそれが何時終了したか及びその終了は正常であったかを決定し、この終了情報をその演算のタグと共に前記維持装置に通信する装置と、

未済演算の表示を機能ユニットに通信する装置と、

機能ユニットからの終了情報に応答して、正常に終了した演算を順番に引退させる装置と、

所与の未済演算が異常に終了したことの情報に応答して、少なくともこの所与の未済演算と、より遅い全ての未済演算とを流出させることを機能ユニットに命令する装置と、

機能ユニットに組合わされ、命令装置によって指定された全ての未済演算を流出させる装置と、

流出した最も早い未済演算のタグに組合わされていたタグから始まるさらなるタグの割り当てを前記タグ割り当て装置に開始させる装置

を具備することを特徴とするコンピュータプロセッサ。

３０．未済タグの表示が最古の未済演算のタグである請求項２９に記載のコンピュータプロセッサ。

３１．未済演算の少なくとも若干をそれぞれが実行できる複数の機能ユニットを含むコンピュータプロセッサにおけるパイプライン化演算を制御する方法であって、

割り当てられたタグを検査することによって２つの未済演算の相対年齢を決定できるように順序を付けたタグの集合の一員であるタグを各未済演算に割り当てる段階と、

所与の未済演算が完了する時点を決定する段階と、

未済タグの独特さを保証するように未済演算の数を制限する段階と

を具備することを特徴とする方法。

３２．上記制限する段階が、一時に多くともｎ演算を未済可能とすることを許容し、

上記タグが２ｎより大きいか等しいある範囲に亘って順次に発行され、

２つの未済演算の相対年齢をそれらのタグの符号付き比較によって決定できるようにした

請求項３１に記載の方法。

３３．機能ユニットが進行する何らかの異常状態を表明する段階と、

所与のタグより遅く発行されたタグを有する全ての未済演算を流出させる段階と

をも具備する請求項3-1に記載の方法。

３４．群内の何れかの演算の流出がその群内の全ての演算の流出をもたらすように関連する演算をグループ化する段階

をも具備する請求項３３に記載の方法。

３５．上記演算の少なくとも若干がが分岐演算であり、

未済分岐演算の結果を予測する段階と、

未済分岐演算の誤った予測を検出する段階と、

誤って予測された分岐演算の結果として発行された全ての未済演算を流出させる段階

をも具備する請求項３１に記載の方法。

３６．最古の未済演算を決定する段階と、
未済演算と引退させる演算との間の境界をマークするタグを挿個することによって、演算の引退が成功したことを上記機能ユニットの少なくとも若干へ通知する段階
をも具備する請求項３１に記載の方法。

３７．群内の全ての演算の引退が可能になった時にのみその群内の何れかの演算の引退が遂行されるように投機演算をグループ化する段階
をも具備する請求項３６に記載の方法。

３８．少なくともそれらの遅演算が未済ではあるが引退はしていない期間の間メモリ書き込みを緩衝する段階と、
それらの遅演算が流出した時に、緩衝されている書き込みを流出させる段階と、
それらの遅演算が引退した時に、緩衝されている書き込みのキャッシュまたはメモリへの配置を完了させる段階
をも具備する請求項３６に記載の方法。

３９．書き込みデータが流出されるか、またはキャッシュもしくはメモリ内に配置される前に、緩衝されている書き込みデータを後続の読み出し演算へ戻す段階
をも具備する請求項３８に記載の方法。

４０．上記発行段階がｍ個のプログラマ可視のレジスタを含む命令集合アーキテクチャを有する入力命令に応答し、少なくとも若干の演算がこれらのレジスタの１つを変更し、
未済のレジスタ変更演算の数をｎに制限する段階と、
少なくとも（ｍ＋ｎ）個の物理レジスタを準備する段階と、
プログラマ可視のレジスタを物理レジスタへ写像する段階
をも具備する請求項３１に記載の方法。

４１．物理レジスタを変更した演算が成功裏に引退してしまうまで物理レジスタが再使用されないことを保証する段階と、
仮想から物理への写像を異常状態が検出された時の先行状態に復元し、従ってプログラマ可視のレジスタの内容を復元する段階
をも具備する請求項４０に記載の方法。

明　細　書

コンピュータの分散型パイプライン制御装置及び方法

発明の背景

本発明は、一般的にはコンピュータに関し、具体的にはコンピュータの効率的なパイプライン制御に関する。

複雑命令集合コンピュータ（ＣＩＳＣ）アーキテクチャの単サイクル実現には深いパイプラインが必要である。ＣＩＳＣアーキテクチャによって直接支援されている複雑な特徴及び保護検査及び強力なメモリ管理システムに、普通のパイプライン技術を組合わせると極めて複雑になる。現在の技術ではパイプラインは多重チップ境界交差の効果を含まなければならない。これらの交差を可能な限り多く排除すべく高レベルのＶＬＳＩ実現が選択されている。システムが比較的少数のデバイスしか含んでいないと、全ての目的のための専用パスを走らせるのに充分な信号ピンは存在しない。これはパスを多目的に使用しなければならないこと、従って集中制御及びスケジューリングメカニズム設計プロセスが極めて複雑になることを意味している。

発明の概要

本発明は、プロセッサ内の機能ユニット全体に分散するパイプライン制御システムを提供する。各ユニットは、それ自身のインタロック及びパイプラインタイミングを限定する。このタイミングは、集中制御装置内で正確に監視することはない。機能ユニットは自律性であるので、自分以外の全てのユニットが各命令をどのように処理するのかの詳細を正確に知る必要がなく、パイプラインタイミングの複雑なシミュレーションが大幅に減少する。本発明は、発生させてはならない機械状態に対する変更のバックアウトを可能ならしめることによって、パイプラインの分散制御を支援する。本発明は、複雑な特別なパイプライン制御論理ではなく一般化された技術を使用し、それによってパイプラインの正しい動作をより有効ならしめている。不要の変化をバックアウトする能力と組合わされた分散制御によって順不同な実行、ペナルティサイクル、及び機能ユニット内及び機能ユニット間の命令の並列処理の領域における性能に重要な長所を得ることができる。これらの能力を実現するための付加的なコスト及び複雑さは極めて僅かである。

詳述すれば、デコーダ論理は各々が対応付けられたタグを有する疑似演算（ p-op またはp-ops ）を、独立的に p-opsを実行できる複数の機能ユニットに発行する。任意時点にはｎまでの p-opsを未済とすることができる。タグは、２つの未済 p-opsの相対年齢（時間）を決定できるようにするために順次に発行される。特定の実施例においては、タグは、少なくとも２ｎの範囲に亘って発行され、リサイクルされる。この範囲は、単純な演算によって相対年齢を決定可能ならしめるのに充分である。この実施例では、１６タグが発行され、７ p-opsを未済とすることが許される。

未済 p-opsは、それらの発行順に引退する。p-ops は、それが完了した時にのみ、即ち通常は全関連機能ユニットによって

それが終了させられた時にのみ引退することができる．若干の場合には、通常ならば引退資格を有する完了した p-opsが、1またはそれ以上の後続する若い p-opsも完了するまで未決に保たれる．最古の未決 p-opsのタグが機能ユニットへ通信されるので、各ユニットは機械の状態を取り消し不能なように変更できるようになった時点を決定できる．

未決 p-opsは、もしそれが機能ユニットによって異常に終了させられれば、打ち切られる．古い p-opsも、もしそれらの引退が異常に終了するp-opが成功裏に完了することを条件としているのであれば、打ち切ることができる．打ち切られる最古の未決 p-op のタグは、機能ユニットへ通信される．これによって、予期せざるプログラムの迂回及び機械が迂回点へ戻された場合に実行の打ち切りを可能ならしめる．

m個のプログラマ可視（仮想）レジスタが存在し、またnまでのレジスタ変更用 p-opsが未決であることを許される命令集合アーキテクチャの場合には、少なくとも（m＋n）個の物理レジスタが設けられる．仮想レジスタを物理レジスタ内に写像（マッピング）するメカニズムが設けられている．この写像は仮想レジスタを変更する各 p-op の宛先としてそれまでに未使用の物理レジスタを使用するために変更されるので、古い仮想レジスタの値をそれが以前に写像されていた物理レジスタ内に保持することができる．写像内に置換された物理レジスタを順番に再使用するのであれば、ある物理レジスタを再使用しなければならない時まで、ある仮想レジスタへ写像されていたp-opが引退または打ち切られるであろうことを保証するための充分な物理レジスタが存在している．仮想対物理写像を規定するポインタの集合及び使用可能なレジスタのリストがn個の最も新しく発行された p-ops群に維持されるので、未決の p-opsを打ち切って、レジスタ間にデータを移動させることなく仮想レジスタを先行値に戻すことができるようになる．

プロセッサの状態を戻すことを可能ならしめる別の技術は、書き込み待ち行列の使用を必要とする．少なくとも発信 p-op（アドレス及びデータを生成する p-op ）が未決である期間中に、書き込み予約待ち行列バッファがメモリまたはデータキャッシュへ書き込む．処理が、メモリ書き込みのバックアウトを必要となり得る点を通過した時だけ、書き込み予約待ち行列エントリをメモリへ出力する．もし発信 p-op が打ち切られれば待ち行列エントリは待ち行列から削除される．若い読み出し p-op が、未決書き込み p-op によって書き込まれるメモリ位置へのアクセスを探索する場合には、書き込み予約待ち行列内に記憶されているデータが読み出し p-op へ供給される．もし書き込み p-op が引退すれば読み出し p-op は、その引退を待機することなく正しいデータを取得している．反対に、もし書き込み p-op が打ち切られれば若い読み出し p-op も打ち切られ、機械状態は書き込み前の点へ成功裏に戻される．

本発明の本質及び長所の更なる理解は、以下の説明及び図面を参照することによって実現されよう．

図面の簡単な説明

図１は、本発明を組み入れたコンピュータシステムのブロック線図、

図２は、デコーダ（ＤＥＣ）の高レベルブロック線図、

図３は、ＤＥＣの詳細なブロック線図、

図４も、ＤＥＣの詳細なブロック線図、

図５も、ＤＥＣの詳細なブロック線図、

図６Ａ～Ｂは、特定シーケンスの追跡を示すブロック線図、

図７Ａ～Ｂも、特定シーケンスの追跡を示すブロック線図、

図８は、レジスタ再割当てを示す概要図、

図９は、メモリ及びキャッシュ制御装置（ＭＣＣ）のブロック線図、

図１０は、整数実行ユニット（ＩＥＵ）のブロック線図、

図１１は、図１０の整数実行ユニット（ＩＥＵ）のブロック線図の続き．

表の簡単な説明

表１は、p-opバスフォーマット、

表２は、物理アドレスバス（PAdeBus)フォーマット、

表３は、データキャッシュバス（DIOBus）フォーマット、

表４は、データ交換バス（DXBus)フォーマット、

表５は、IEU 終端バスフォーマット、

表６は、AP終端バスフォーマット、

表７は、p-op発行及び終結のシーケンス、

実施例

システムの概要

図１は、本発明を組み入れたＣＰＵ　１０のブロック線図である．Ｆ８６と呼ばれることもあるＣＰＵは、カリフォルニア州サンタクララのインテル・コーポレーションから1986年に刊行されたIntel 80386 プログラマーズリファレンスマニュアルに記載されているIntel 80386 の命令集合と互換性のある命令集合（マクロ命令）を実行するように設計されている．図中の各ブロックは、一般的に現在具体化されている分離した集積回路チップまたはチップ群に対応する．ＣＰＵは、システムバス１１を介してメモリ制御装置、Ｉ／Ｏデバイス、及び多分他のＣＰＵのような外部デバイスと通信する．機能ユニットの下部に示す参照番号は、これらの外部デバイスではないＣＰＵ１０内の要素を意味するものと理解されたい．

命令デコーダ（ DEC）１２は命令取り込み、命令デコード及びパイプライン制御を遂行する．ＤＥＣ　１２は、３つまでの同時命令の流れの命令先取りを任意選択的に交互配置する．ＤＥＣ　１２は、完全に連想型の分岐予測キャッシュ（BPC）１３を含む．ＢＰＣは編成された構造であり、分岐履歴データ、物理分岐目標アドレス、及び分岐目標バッファを各キャッシュメモリ毎に含む．分岐命令がデコードされると、ＢＰＣはその分岐に関する情報を調べる．予測される方向には無関係に分岐は単一のサイクル中に実行され、パイプラインバブルを生じさせることはない．

各サイクルに、ＢＰＣ内の３つの命令バッファまたは分岐目標バッファの1つからマクロ命令が選択される．このマクロ命令はデコードされ、疑似op（p-op）とも、または命令もしくは演算とも呼ばれることがある内部９６ビットデコード済命令語にアセンブルされ、各種機能ユニットへディスパッチされる．命令のデコードは、一般に単一サイクルレートで進められる．

ＤＥＣ　１２が発行する各p-opには、機械内で現在未済の各p-opを独特に識別するタグを与えられる。タグは昇順で発行され任意の２つのタグの相対年齢を容易に決定できるようにしている。チップ間のバストランザクションは発信 p-op のタグを含む。機能ユニットは、p-op、アドレス、及びオペランドとこれらのタグとを組（対）にする。

ＤＥＣ　１２はまた未済p-opのステータスの追跡と、パイプラインの制御と、必要に応じての例外処理の呼出しとに責を負っている。

アドレス準備ユニット（AP）１５は実行アドレスを計算し、セグメント再配置を遂行し、要求時ページングされるメモリ管理システムを実現する。ＡＰは変換索引バッファ(TLB) を含む。

整数実行ユニット(IEU) １７は殆どの整数命令の単一サイクル実行を遂行する。ＩＥＵは、８×３２乗算器及び累算器アレイ、並びに乗算命令及び除算命令のマイクロコードを含む。パイプライン制御アーキテクチャは、ＩＥＵの整数命令の並列実行及び順不同の実行の両方または何れか一方の遂行を可能ならしめる。

数値プロセッサ（NP）２０は、任意選択的に、ＣＰＵ内に含ませることができる。これはIEEE浮動小数点標準を高性能に実現する。ＮＰはパイプライン内に装備され、命令及びオペランドの転送に関して何等の特別なオーバヘッドも経験しない。整数（IEU)及び浮動小数点（NP）命令は同時に遂行される。

メモリ及びキャッシュ制御装置（MCC）２５は、命令及びデータキャッシュを制御する責を負い、キャッシュコヒーレンシープロトコルを実現する。ＭＣＣはシステムバス１１へのインタフェースを制御して、キャッシュとメモリとの間の高速単一及びブロックモード転送を支援する。後述するように、ＭＣＣは、整数、浮動小数点、及びシステム書き込みのための書き込み予約表をも含み、またリードアフタライト短絡回路経路を含む。

命令キャッシュサブシステムは、タグＲＡＭチップ（ITAG）２７及びキャッシュＲＡＭチップ（ICACHE）３０を含む。ＩＴＡＧ　２７内の各エントリは、ＩＣＡＣＨＥ　３０内の対応する線のためのアドレスタグ、有効ビット、及びアテンションビットを含む。アテンションビットは、ＤＥＣチップもＢＰＣ内にキャッシュされたこの線からのデータを有することができることを指示する。ＩＴＡＧ　２７は、命令流アドレスレジスタ３１の集合をも含み、各レジスタは３つのどうあっても未済の流れの１つ１つに対応付けられた取り込みアドレスを含む。

データキャッシュサブシステムは、タグＲＡＭチップ(DTAG)３２及びキャッシュＲＡＭチップ（DCACHE）３５を含む。ＤＴＡＧ　３２は、ＤＣＡＣＨＥ　３５内の各線のためのアドレスタグ及び線状態ビットを含む。考えられる線状態は、欠落、共用読み出し、オウンドクリーン、及びオウンドダーティーであり、ライトバックマルチプロセッサキャッシュコヒーレンシープロトコル（変更された書き込み１度）を支援する。タグＲＡＭはデュアルポート型であり、単一のサイクル中にＣＰＵ及びバスの両者がキャッシュルックアップをスヌープすることを可能ならしめる。データキャッシュインタフェース（DCI)チップ３７はＤＣＡＣＨＥ　３５をシステムバス１１へインタフェースする。

各機能ユニットは、電力及び接地プレーン、並びに組合わされた線結合コンデンサを含む特注のセラミックＰＧＡ内にパッケージされている。ピンのほぼ２５％は電力及び接地に当てられている。０．８ミクロン乃至１．２ミクロンプロセスの場合Ｉ／Ｏ遅延はオンチップ限界経路と対等である。チップ間Ｉ／ＯはパイプライＮ内に組み込まれているので機械にサイクル時間を付加しない。ＩＣＡＣＨＥ　３０及びＤＣＡＣＨＥ　３５は普通のスタティックＲＡＭを使用している。

種々の機能ユニット間の通信は多数の内部バスを介して遂行される。これらには、命令取り込み用６４ビットＩＦＥＴＣＨ＿ＤＡＴＡバス５０、発行された p-opsをＡＰ、ＩＥＵ、ＭＣ及びＮＰへ通信する１０４ビットp-opバス５２、未済 p-ops情報をＡＰ、ＩＥＵ、ＭＣＣ及びＮＰへ通信する５ビットタグ状態バス５３、物理アドレスを通信する３２ビット物理アドレスバス（PAdrBus)５５、データキャッシュ転送用６４ビット（各方向に３２ビット）データキャッシュバス（DIOBus)、チップ間交換用３２ビットデータ交換バス（DXBus)５８、キャッシュ／メモリ更新用６４ビットバス、及び種々の終了バス（即ち各機能ユニットからＤＥＣ　１２までのＡＰ終了バス６０、ＩＥＵ終了バス６２、ＮＰ終了バス６３、及びＭＣＣ終了バス６５）が含まれる。これらのバスの若干は全幅であり、若干は半幅（時間多重化）である。一般的に、種々ユニット間の対話は内部プロセッサバス上に充分に限定されたトランザクションに制限される。

種々のこれらのバスの詳細に関しては後述する。標準ＣＭＯＳスタイル時間多重化Ｉ／Ｏの使用方法によれば、転送はシステムクロックのフェーズ１（φ１）とフェーズ２（φ２）との間の境界で発生することを暗示している。φ２転送は、送信チップがφ１の終りの前に有効データをその１／Ｏドライバへ準備する必要がある。有効データは後続φ２中に受信チップの１／Ｏ受信器によって供給される。φ１転送は丁度反対のタイミングである。

表１～６はそれぞれ、p-opバス５２、ＰＡｄｒＢｕｓ５５、ＤＩＯＢｕｓ５７、ＤＸＢｕｓ５８、ＩＥＵ終了バス６２、及びＡＰ終了バス６０のバスフォーマットを示す。

パイプライン制御システムの概要

プロセッサのパイプライン制御は上述の種々ユニットにまたがって分散している。パイプラインの集中スケジューリングまたはスコアボーディングは遂行されない。ＤＥＣ　１２はアーキテクチャ内の若干の総合資源制約を観測し、資源制限を犯すp-opの発行を適時遅らせる。各機能ユニットは、それ自身の内部操作をスケジュールする責を負う。インタロック検査はローカルレベルで遂行される。

深くパイプライン化された機械では、パイプラインの種々の段階における例外検出が制御に重大な困難をもたらす。各段階は、他の段階が未だに先行命令の例外を検出できる間は、状態

の変更を通らせるに当たって注意深くなければならない。専用制御論理が一般的であり、パイプラインシミュレーションを注意深く遂行しなければならない。

プロセッサは、単純で、一般的で且つパワフルな幾つかの技術を使用してこの複雑さを処理する。ＤＥＣ　１２はデコードされた命令（ p-ops）を発行し、機能ユニットは他の機能ユニットによる例外の検出の結果には拘りなくアドレス及びオペランドを処理する。前述のように、各 p-op にはそれが発行される時にＤＥＣ　１２によってタグが割り当てられており、ＤＥＣはこのタグを使用して p-op を追跡する。

ＤＥＣ　１２は、実行が例外の点を過ぎて遂行した時点を決定する責を負う。以下に説明する技術を使用してＤＥＣは機械の状態を、例外を生じさせた p-op の直前の点（障害例外）または後続点（トラップ例外）に復元する。

上述したように、各機能ユニットはＤＥＣ　１２へ戻る終了バスを有している。これらのバス上の信号は（タグによって）p-opが完了した時点と、そのユニットによってどの例外（もしあれば）が検出されたのかを指示する。ＤＥＣはこの情報を使用して、機械内でどの p-opsが未済であるかを追跡し、資源割当を追跡し、そして例外処理を開始しなければならない時点を決定する。

異常終了に応答してＤＥＣ　１２は、機械の状態を例外の点へ戻し、例外ハンドラを呼出して異なる命令流またはマイクロ命令のシーケンスの何れかを発行し始める。プロセッサは５つの一般的メカニズムの１またはそれ以上を使用して、異常終了へのＤＥＣの応答の部分としての特定状態へ機械を戻すことができるようにする。これらは打ち切りサイクルの発行、レジスタの再割り当て、書き込み予約表の使用、履歴スタックの使用、及び機能ユニット直列化である。

打ち切りサイクルは、ＤＥＣ　１２が発行した命令を機械から一掃しなければならない時にＤＥＣが発行する。打ち切りサイクル中に、完了することを許すべき命令と機械から追放しなければならない命令との間の境界を識別するタグが全ての機能ユニットに供給される。

レジスタ再割り当ては、一般的レジスタファイル及びセグメントレジスタファイルの状態を復元し、打ち切らなければならない命令のために行われた変更を流出させるために使用される。機能ユニットは、命令集合が指定するよりも多くの物理的に使用可能なレジスタを有している。ＤＥＣ　１２は、プログラマ可視（もしくは仮想）レジスタを物理レジスタへ写像するポインタの集合を維持している。デコードされた命令をアセンブルするに当たってＤＥＣは、適切な物理レジスタ番号をレジスタ指定フィールドに置換する。

仮想レジスタを変更する場合、ＤＥＣは先ず新しい物理レジスタを割り当て、ポインタ集合を変更し、割り当てられたレジスタ番号を宛先レジスタとして使用する。命令の実行の後でも古い物理レジスタは未だ仮想レジスタの変更された値を含んでいる。レジスタ変更をバックアウトするためには、ＤＥＣはポインタ集合を命令発行前の値に復元しなければならない。

解放された物理レジスタは、物理レジスタの内容を必要としなくなった後までそれらが自由リストの先頭に現れることがないように充分に長くした自由リストの終りに配置される。ＤＥＣは、以下に説明するように、ポインタ値の履歴スタックを維持している。

書き込み予約表はＭＣＣ　２５において使用され、その書き込みを打ち切ってはならないことが知られるまでデータ書き込みを待機させる。ＭＣＣは内部データバス上のアドレス及びオペランドを受信し、それらをタグによって突き合わせ、そのようにしても安全である場合に不可逆書き込みを遂行する。

履歴スタックは、レジスタ再割り当てポインタ、フラグレジスタ、及びプログラムカウンタのような雑機械状態を保管及び復元するために使用される。

稀にしか変更されない機械状態の場合、値の履歴スタックのコストは無視される。これらの場合、変更を遂行する機能ユニット（そしてそのユニットだけ）が処理を停止し、機械内の最古の未済命令のタグ（ＤＥＣから供給される）が各サイクルに調べられて機械内の全ての古い命令が成功裏に完了した時点が決定される。この時点になると機械状態の古い値を予約する必要はなくなり、また機能ユニットは機械状態の不可逆な変更を行う。

状態変更をバックアウトする能力と組合わされた分散型パイプライン制御スキームは、多くの性能最適化を可能にする。

各機能ユニットは全ての p-opsを受信できるが、実際にそのユニットにおいて処理を必要とする p-opsだけを処理する。これは、段が有用な作業を行うと否とに拘らず命令が全ての段を通って流れる普通のパイプラインとは対照的である。

更に各ユニットは、全ての入力オペランドが使用可能になると直ちに演算を遂行する。直ちに実行する準備が整っていない p-ops はそのユニットの p-op待ち行列内に記憶される。完了すると、その結果はさらなる処理を行うために次の段に渡され、次の演算が調べられる。１つの段は、その段が実行するために使用可能な何ものをも有していない場合に限って実行を停止する。

この挙動によって機能ユニット間で順不同の実行が可能になる。例えばアドレス生成インタロックを有するメモリ書き込みの場合、ＡＰはメモリアドレスを計算することはできないであろう。しかしＩＥＵはデータを供給することができ、直ちにそれを行い、その後次の命令へ継続する。ＡＰのインタロックは他のパイプライン段内にパイプラインバブルを作成する必要はない。後刻、ＩＥＵは乗算の遂行を通らせるか、またはメモリオペランドを待機することができる。この時点でＡＰはＩＥＵに追いつく機会を有する。

特定の機能ユニットの観点からすれば、これは複雑な概念ではない。機能ユニットは局部的に決定を行い、それが命令を完全に順不同ならしめているかも知れないことに全く気付かない。パイプライン制御メカニズムは、順不同に実行された命令によって行われた変更を流出させ得ることを保証する。機能ユニットは特別な検査は行わない。

機能ユニット間の順不同な実行は、プロセッサ内で行われる分散した決定の結果として自由に発生する。１つの機能ユニッ

ト内においてさえ、命令は安全に順不同で実行し得る。ＩＥＵ１７は、この内部順不同実行の例を提供する。ＩＥＵはその命令待ち行列の先頭を調べて、その実行準備が整っているか否かを見出す。もしデータインタロックが直ちに実行することを阻止していれば、ＩＥＵは次に若い命令を調べて、その実行準備が整っているか否かを見出す。このプロセスは実行できる命令を見出すまで続けられる。ＩＥＵは、実行の準備が整っている使用可能な命令が存在しない場合に限ってデータインタロックペナルティを支払う。

たとえＩＥＵがインタロックペナルティを払ったとしても、それはプロセッサが全体として１つのサイクルを失ったことを意味するものではない。たとえＩＥＵが遅れたとしても、後にＩＥＵを必要としない命令が発行された時に追いつくことができる。最後に、１または複数のペナルティサイクルは、ＡＰ１５からの１または複数のペナルティサイクルと重なり合うことができる。

命令を順不同に実行することを選択する機能ユニットの特別な場合は、機能ユニット内における命令の並列実行である。即ち、この概念は、複数のサイクルを要する命令に適用される。他の単一サイクル命令の並列実行は、多重サイクル命令が１サイクルの実効スループットを有することを可能にする。

ＤＣａｃｈｅミスは、通常は全キャッシュミスペナルティのためにパイプラインを停止させる。機能ユニットがキャッシュデータを用いずに実行できる演算を見出すことができる範囲までキャッシュミスペナルティは減少される。このことはＡＰチップのＴＬＢにおけるミスに関しても真である。これらの場合は、ペナルティサイクルの数が通常かなり高くそれらを有用作業に完全に重ね合わせることが困難な他の場合とは異なる。

擬似Ｏｐバスフォーマット

表１に p-op バス５２のフォーマットを示す。このバスは５２ビット幅であり、時間多重化されたバスである。ＤＥＣ１２は単独でこのバスを駆動して p-opsをＡＰ、ＩＥＵ、及びＮＰへ発行する。バスは標準ＣＭＯＳスタイル時間多重化Ｉ／Ｏを使用する。

典型的には、１つの３８６／３８７マクロ命令はＤＥＣによって関連機能ユニットへ発行される１つの p-op に変換される。若干の場合には、１つのマクロ命令が発行済 p-opsのシーケンスをもたらす。この p-op 発行シーケンスはアトミック(atomic)であり、即ち１つのマクロ命令の p-opsの発行が別のマクロ命令の p-opsの発行（または例外処理シーケンス）と交互配置されることはない。

典型的なマクロ命令の場合、１つの p-op は、全ての関連機能ユニットにそのマクロ命令の必要演算を進行可能ならしめるのに充分な情報を含む。これは、メモリオペランドアドレス計算及びセグメント、発信及び宛先オペランドレジスタ、ＡＬＵ演算、オペランドサイズ、オペランド経路指定、ステータスフラグ変更、及び p-op タグ、並びに関連した変位及び即値データ値の両方または何れか一方の指定を含む。ＮＰ p-opsもマイクロアドレスを指定する。

殆どの p-opsは、１クロックサイクル中に両クロックフェーズ（φ１及びφ２）を使用して p-op バス上に転送される。φ１は p-op 内に含まれる殆ど全ての制御情報を転送するために使用され、φ２は変位及び即値の両方または何れか一方を（制御情報の僅かな推特別ビットと共に）転送するために使用される。変位及び即値の両方を含む p-opsの若干の場合（５２ビットにパックすることができない）には、即値を転送するために第２クロックサイクルが使用される。この第２サイクルは常に第１クロックサイクルの直後に続く。変位は第１サイクルのφ２に転送され、即値は第２サイクルのφ２に転送される。

ＤＥＣ　１２は、全てのクロックサイクル中に p-op バスを駆動する。通常はこれは正常 p-op であるが、ＤＥＣが正常p-opを発行する準備が整っていないか、または発行できないサイクル中は、ＤＥＣは代わりに空 p-op を送る。

P-op内に情報をエンコードすることのフィロソフィは、何よりも先ずあるクロックサイクル内の可能な限り早い時点にエンコードされていないか、または迅速にデコードできる形状で制御情報を供給することである。これは特に各機能ユニットにおける遅さが臨界的な演算の開始に関して、及び変位及び即値の抽出と適切なアドレス及びデータオペランドの導出とに関して真である。それ程臨界的ではない制御情報だけがφ２中に転送されるが、一般的にはφ２中には各機能ユニットはレジスタ及びp-opの両方からのオペランドのアセンブル／取り込みを行って、次のφ１に各機能ユニットが内部計算等を開始できるようにすべきである。

前述のように殆どのマクロ命令は単一の p-op に変換される。これは若干のより複雑なマイクロ命令を含み、この複雑さはマイクロ命令を介して機能ユニットの１つにおいて処理（例えば、ＩＥＵ、ＡＰ内のＰＯＰＡにおける乗算）されなければならない。しかし可能な場合には、複雑なマクロ命令は、総合的なシーケンスには気付くことなく機能ユニットによって独立的に実行される p-op シーケンスに変換される。若干の場合には、例えば複数のレジスタ再割り当て(p-op 当たり１つだけが許される）、適切なメモリ要求生成のためにＡＰが要求する複数の p-op タグ、またはＡＰによる複数のレジスタ及びフラグの更新をＡＰに通信する必要がある制御情報の量または本質のために、 p-op シーケンスが本質的に必要である。

若干の複雑なマクロ命令の場合、上述の組合わせも発生し得る。即ち、 p-op のシーケンスが発行され、機能ユニットの１つがマイクロコード内へ進んでマクロ命令のコア部分または全部を後続する p-opsと共に実行する。例えばシーケンスの最初の p-op がＡＰ及びＩＥＵによって処理され、別にＡＰはマイクロコード内へ進んでさらなる演算を進行する。これらのさらなる演算は発行される後続 p-opsに対応する。概念的には、シーケンスの p-opsは機能ユニットによって独立的に実行され、この場合にはこれはＩＥＵに関して文字通り真である。しかしマクロ命令の本質のために、ＡＰは p-op シーケンスを大域的に知る必要がある。従ってこの場合、ＡＰはマイクロコード内へ進み、単純に後続 p-opsと同期する。外観ではＡＰは各 p-op を独立的に実行し終了するが、内部的にはＡＰは p-op

タグと各 p-cp の1または2フィールドだけを使用するのである。

機能ユニットによる p-opsの発行と認識に関して一般的な性質の2つの付加的な説明をしておく。第1に、殆どの p-opsは全ての機能ユニットによるそれらの p-op 入力待ち行列内への待ち合わせをしない。その結果、各機能ユニットは、全てのp-ops を見ず、処理せず、または時間を消費しない。一般的な場合には、 p-op は、ＡＰ及びＩＥＵによって、またはＡＰ及びＮＰによって認識される。若干の p-opsはＡＰだけが見ればよく、1または2 p-opsは3つの全機能ユニットによって認識される。ＡＰだけが全ての p-opsを見るのである。

第2に、ＤＥＣが例外処理に入るある理由が存在する場合にはＤＥＣはそのようにし、p-ops に関連するより新しい例外処理の打ち切りを要するかも知れない未済先行 p-opsが未だに存在していても、対応付けられた p-opsを発行する。一般に、ＤＥＣはマクロ命令の観点から適切な演算を保証するように、p-ops 発行に当たって最低必要な自制を遵行する。

関連点は、微視的観点から（即ち個々の p-opsのレベルにおいて）、ＤＥＣが発行できる p-op シーケンスに対して、またはそれらの発行のタイミングに対して極めて僅かな見掛け上の制約が存在し、従って機能ユニットが僅かな仮定をなし得ることである。これは特に、p-ops の打ち切りに関して僅かな仮定をなすことができるという事実に適用される。許される未済 p-opsの最大全数及び許される未済ＮＰ p-opsの最大数、及び任意時点にどの p-opsが活動／未済であり得るかに関する保証のような最も基本的な制約だけが明白である。

適切な巨視的命令実行を保証することに関して、簡単に説明する価値がある1つの面が存在する。若干の p-opsは、Ｆ８６マイクロアーキテクチャが p-op による変更後にバックアウトする能力を支援しないプログラマ可視状態を変更する。概念的には、これは、 p-opsが全て実行される前にその p-op を永続的に実行することをＤＥＣが保証できるように、機能ユニットのある程度の静止を必要とする。これは、全ての機能ユニットが静止状態に到達してしまうまでＤＥＣがその p-op （及び全ての後続 p-ops）の発行を遅らせるという一般的な技法では行われない。その代わりとして、所与の p-op に対して静止を必要とする各ユニットだけによる局所化された（機能ユニット）基準で行われる。ＤＥＣは関連機能ユニットによって必要な程度の静止を遂行しながら、これと後続する p-opsとを発行することができる。更に、静止内に含まれないユニットは、後続p-ops を完全に実行し続けることができる。

ＤＥＣの概要，擬似Ｏｐ追跡，及び発行制御

各擬似op（p-op）がＤＥＣから P-Op バス上に発行されると、それは適切な機能ユニット（ＡＰ、ＩＥＵ、ＮＰ）によって待機させられる。次いで各機能ユニットは他のユニットにゆるく結合されている p-op の流れを処理し、各 p-op が完了するとＤＥＣ　１２へ終了を通知する。図2にブロック線図で示すＤＥＣ１２は、フロントエンド１００、デコーダ１０２及びバックエンド１０５からなる。図3はＤＥＣフロントエンドを、図4はＤＥＣデコーダを、図4はＤＥＣバックエンドを示す。

ＤＥＣフロントエンド１００は命令バイトを取り込み、デコーダへ供給する責を負う。命令はＢＰＣ　１３から、またはＩＦＥＴＣＨ＿ＤＡＴＡバス５０によって供給されている3つの命令バッファの1つから供給される。命令バイトは、命令をＰＣ（プログラムカウンタ）レジスタ１１２からの情報に基づいて位置合わせする回転／けた移動論理１１０へ供給される（一時に２４バイト）。8バイトがデコーダ１０２へ供給され、デコーダ１０２は命令長を決定してそれをＰＣ論理１１２へへ通信する。命令が8バイトより長い場合には、1サイクルに8バイトが通信され、8バイトを超える命令は次のサイクルに通信される。

フロントエンド論理１１５は流れスクック１１７を制御し、流れアドレスをＩＴＡＧ　２７へ供給する。2つまでの未済分岐が、従って3つの未済の流れが存在可能である。制御論理はどの流れを取り込むのかを指定する命令要求をＩＴＡＧ　２７内の命令流アドレスレジスタ３１へ発行し、流れを識別する有効ビットを受信する。ＩＴＡＧがアドレスを供給すると、それは適切なアドレスレジスタをインクリメントさせる。制御論理１１５は、自己修飾コードに関する命令の流れ内への書き込みを検出するＰＡＤＲ監視論理１２０からの信号も受信する。

ＤＥＣデコーダ１０２はマクロ命令をデコードし、全てのp-opシーケンスを p-op バス５２上に発行する。デコーダは、命令レジスタ１３０内にロードされている命令バイト（マクロ命令）をフロントエンド１００から受信する。マクロ命令はデコード論理１３２によってデコードされ、 p-op 型デコード論理１３５はp-op型に関する情報をフロンテンド及びバックエンドへ送信し、一方命令長デコード論理１３７はフロントエンド内のＰＣ論理１１２と通信する。

デコーダ p-op アセンブリ論理１４０はデコード論理１３２から p-op を受信し、バックエンドからのレジスタ割り当て情報に従ってそれらを変更する。p-opsは、 p-op バス５２上に駆動されると p-op 出力待ち行列１４２内にロードされる。発行は、バックエンドからの制御信号に基づいて発行保持論理１４５によって遅らされる。

デコーダ１０２は、多重 p-opsが単一マクロ命令を発生する場合に発行を制御するシーケンサ１４７を含む。デコード保持論理１５０は、フロントエンドから有効命令バイトが到来しない場合に処理を阻止する。 p-op の発行に伴ってデコーダ１０２はタグを割り当てる。タグは循環シーケンスで発行され、従って再使用されるが所与の時点には1つの p-op だけがそのタグに対応付けられている。タグの範囲は、相対年齢を決定できるように、未済が許される p-op の数に対して充分に大きくなければならない。未済 p-op の最大数の少なくとも2倍の範囲とすれば単純な推算によってこのような決定が可能になる。

バックエンド１０５は、全ての未済 p-opsがＣＰＵの周囲に浮動するのを追跡し続ける。信頼できる動作（ p-op 、アドレス、及びデータ処理を制御するＣＰＵのタグ付けスキームに関連する）を保証し、機能ユニット終了によって知らされる異常

状態を調停し、そして適切な動作を開始するように p-opsの発行を適切に制御する必要がある。デコーダが p-op を発行すると、それはその p-op に関する情報と共にバックエンドに送られる。これは上述のタスクを遂行するために必要な正しい動作を識別するのに使用される。

バックエンドは、全ての未済 p-opsを追跡し続ける追跡論理１６０と、未済 p-opsに応答しＣＰＵの正しく且つ信頼できる動作に要求される種々の制約（後述）を絶えず満足するようにデコーダによる最後の p-opsの発行を制御する保持条件論理１６５とを含む。追跡論理１６０は、最古の未済 p-op のタグ（ＯＯタグ）を含む情報をタグステータスバス５３へ供給する。バックエンドは、p-ops の打ち切りを処理する打ち切り論理１７０と、後述するポインタ集合アレイ１７７及び自由リストアレイ１７８を維持するレジスタ再割り当て論理１７５と、タグステータスバス５３を制御するタグ生成論理１７９をも含む。

バックエンド終了パス論理１８０は各機能ユニットから終了情報を受信し、追跡論理１６０及び打ち切り論理１７０が各未済 p-op のステータスを維持することを可能ならしめる。若干はある将来時点まで累積される。正常動作中のこの追跡は主として最後の p-opsの発行に影響を与える。しかし対応する終了によって機能ユニットから異常が通知されると、バックエンドは所与の p-op の多重異常終了を解決し、適切な応答を開始する。これは、ＣＰＵの状態を p-op 処理のある先行状態まで戻すように他の全ての機能ユニット（ＭＣＣも含む）へ打ち切りサイクルを送ることを含む。

追跡論理１６０及び打ち切り論理１７０は、全ての未済 p-ops に関する特定情報を記憶するレジスタを含む。これらのレジスタは、未済 p-opsのタグの３最下位ビットに対応する番号０－７を付された８つの同一レジスタ集合として編成されている。多くとも７ p-opsが未済であることができ、またタグが単じに発行されるから、相対年齢は位置番号に基づいて決定することができる。追跡論理１６０は関連論理を有する８つずつの状態レジスタ１９０、終了レジスタ１９２、及び p-op 情報レジスタ１９３を含む。打ち切り論理１７０は関連論理を有する８つずつの応答選択レジスタ１９５、優先順位論理レジスタ１９７、及び終了記憶情報レジスタ１９８を含む。

各状態レジスタ１９０は、その位置に対応するタグを有する p-opが未済であればセットされる単一の状態ビットを記憶する。各終了レジスタ１９２は、機能ユニット当たり１つの終了ビットを記憶する。このビットは機能ユニットが p-opを終了するか、または機能ユニットが p-opに対して動作を起こす必要がない時にセットされる。

各 p-op 情報レジスタ１９３は対応付けられた p-op に関係する８ビットを記憶する。これらは、機能ユニットが操作する p-opのタグの最上位ビット、 p-op の型（例えば浮動小数点、分岐）、分岐予測情報、及び打ち切り群ビットを含む。即ち、"０" は p-op が最終番号ではなく従って単独では引退できないことを表し、一方 "１" は打ち切り群ビットに "０" を有する隣接の古い p-opsを打ち切ることなくその p-op を打ち切ることはできないことを表す。

状態ビットの収集は最古の未済 p-op を識別可能ならしめる。 p-op の位置はタグの３最下位ビットを供給し、情報レジスタは最上位ビットを供給する。状態ビット及び p-op 情報レジスタ１９３内のビットは、後述するように保持条件計算論理１６５が保持条件を決定することを可能にする。

各応答選択レジスタ１９５は、どの応答が必要なのかに関する情報をフロントエンドに供給する。各優先順位論理レジスタ１９７は、所与の p-op の多重異常終了に対する応答に対して取るべき適切な動作を指定する。各終了記憶レジスタ１９８は関連 p-op に作用する機能ユニットからの異常終了の詳細を含む詳細な終了情報を維持する。

打ち切りが発生した場合を除く殆どの場合、機能ユニットは未済 p-opsのステータスに関係はない。これに対する主な例外はＭＣＣ　２５であり、ＭＣＣはキャッシュ内へのメモリ及びＩ／Ｏ書き込み及びシステムの残余への出力の両方または何れか一方を実際に遂行するに当たって安全であることを知る必要がある。特別な場合にはＡＰ及びＩＥＵも若干の p-opsを実行することが安全であることを知る必要がある。これらの要求は全て、ＯＯタグ及び信号打ち切りを表すタグステータスバス５３上に各クロックサイクル毎に連続的に情報を発行するバックエンドによって満足される。

<u>タグステータスバス</u>

タグステータスバス５３は５ビットのバスであり、それらの信号は φ１だけに固定される。殆どのサイクルにおいてそうであるが、ビット＜５＞が０である時には、ビット＜４..０＞は最古の未済 p-op のタグであるＯＯタグを表す。ビット＜５＞が１である時には打ち切りが指示され、ビット＜４..０＞は p-opのタグをもとに戻って打ち切ることを指示する。これは打ち切りタグ（ATag）と呼ばれる。打ち切りサイクル中にバックエンド１０５は、デコーダの次の p-op の発行を無効にし、２つの型の空 p-opsの１つを発行させる。タグステータスバスが、タグ＝ｉを有する p-op が最古の未済 p-op であることを指示している場合には、これは全ての古い p-ops（即ち、４ビットの２の補数演算に基づいてタグ＜ｉを有する p-ops）は最早未済ではなく、引退するものと見做すことを意味する。 p-op(i)を含む全ての若い発行済 p-ops（即ち、タグ≧ｉを有する p-ops）は未済である。勿論これは、発行済で期待に打ち切られる p-opsを除外する。

未済であると見做す p-op は、それは未だ打ち切り可能であることを意味し、実際にこれは p-opsを引退させる時を決定するに当たってバックエンド１０５が使用する演算定義である。全ての機能ユニットによって p-opsの処理が完了すると、（それらの終了に基づいて）可能な限り直ちにそれらを引退させるのが一般である。しかし p-opsを実際に引退させることができるようになった時には種々の制約がある。その詳細の若干を以下に説明する。

最古の未済 p-op を引退させる時、タグステータスバスはこれをＯＯタグ＝ｉ指示からＯＯタグ＝ｉ＋１指示へ前進させることによって表示する。各及び全クロックサイクルに最古の未

済タグを前進させることができる。また○○タグ＝ｉから○○タグ＝ｉ＋ｎ（但し、１≦ｎ≦７）へジャンプさせて１クロックサイクルに幾つかの p-opsを効果的に引退させることもできる。もし未済 p-opsが存在しなければ、タグステータスバスは発行される次のタグを最古の未済として指示する。

タグ＝ｉを有する p-op （p-op (i)）までの打ち切りは、タグ≧ｉを有する全ての p-op （４ビット符号を付けた２の補数演算に基づいて）を流出させ、ＣＰＵの状態を p-op (i-1) と p-op (i)との間にあった時の状態までロールバックさせるべきである。これは次の p-op タグを発行することを含む。換言すれば打ち切りは p-op(i)及び全ての若い p-opsを流出させ、ＣＰＵをこれらの p-opsが見掛け上発行されない状態に復元すべきである。

タグ＝ｉまでの打ち切りは随時発生させることができ、p-op (i) が最古の未済p-opになるまで遅延させる必要はない。またこのような打ち切りはタグ≧ｉを有する p-opsが存在市内場合にも発生させることができる。しかしそれでも、打ち切りタグ及び全ての未済 p-opsのタグは、相対年齢に関する全てのタグ比較が未だに信頼できるものであることを保証される。（傍注として、例えばもし７つの未済 p-opsが存在していてこの打ち切りが発生すれば、打ち切りタグは７番目の（即ち最も若い）p-opのタグより１つ大きくなければならない。）

この流出及び状態のローリングバックは、（概略で）打ち切りが通知されたサイクル中に各機能ユニットによって遂行されなければならない。デコーダ１０２は、その次のサイクルに新しい p-opsの発行を開始するかも知れないから、このことが必要なのである。これは特に、方向または型（遠方への制御の転送のための）が誤予測された「制御の転送」マクロ命令に関して真である。

要約すれば、各機能ユニットは１サイクル中にそれ自身を空にし、そのサイクルの終りまでに処理の状態を正常に戻さなければならない。

一般的に、打ち切りサイクルに続くサイクルには、別の打ち切りサイクルを発生させることができるか、 p-op を発行（より多くの開後のサイクルを用いて）できるか、または単純な空 p-opを発行（デコーダは未だ次の p-op を発行する準備が整っていないから）できるの何れかとなる。ある打ち切りサイクルに続く次のサイクルが別の打ち切りサイクルではないものとすれば、最古の未済を指示する p-op タグは、その打ち切りサイクルに先行するものと同じであっても、または戻って打ち切られたタグまで前進せしめられたタグの番号を有していてもよい。この最後の場合は、打ち切り後に全ての先行（古い）p-opsが引退し、そして勿論全ての若い未済 p-opsが最早存在しない時に発生する。

タグ発行

以下の説明は p-op タグと、それらが何であるかの概要と、ＤＥＣ　１２がそれらをどのように発行するかに関する。全てのタグは発行される全ての p-opsの一部としてＤＥＣから発せられる。各 p-op タグは、各p-opに関連するアドレス及びデータにタグを付けるために機能ユニットによって使用される。７つまでの未済 p-opsが許されるものとすれば、少なくとも３ビットのタグが必要である。これは、相対年齢に関して p-op タグを比較するのを簡易化するように、更に１つの上位ビットを用いて４ビットタグに拡張される。即ち、以下に説明するようにしてタグを割り当てると、４ビットの２の補数符号付き比較が２つのタグの相対年齢を確実に指示する。p-ops を明確に識別するためには任意の時点に３最下位ビットだけが必要であることに注目されたい。

マクロ命令の順番に対して、これらの命令から得られる全ての p-opsは順番に発行され、タグも順番に割り当てられる。１６のタグ値は全て有効タグと考えられ、タグ順は［次のタグ］：＝（［現タグ］＋１）モジュロ１６として定義される。従って相対年齢に関する上記比較は確実に作動する。

打ち切りがない命令処理中は、以上の説明がそのまま適用される。タグ＝ｉまで戻って打ち切りが発生し、ＣＰＵ状態がp-ops(i)の直前までロールバックすると、タグ割り当てもタグ＝ｉまで戻ってリセットされる。相対年齢比較の信頼性を保証し続けるために、ＤＥＣはこの時点からタグ＝１で始まる新しい p-ops を発行しなければならない。効果的に、打ち切られたp-ops のタグが新しい p-opsに再発行される。これは、例えば、先行打ち切りより前の点まで戻った打ち切りが、あたかも第２の打ち切りだけが発生したかのような効果を呈することを意味する。

より一般的には、打ち切りサイクル及び p-op タグ発行に関して仮想的に制約されないシナリオの組合が発生し得る。例えばp-ops(3-7)が未済中であれば p-op(5)まで打ち切り、タグ 5-8を発行し、p-op(6) まで打ち切り、p-op(4) まで打ち切り、タグ 4-5を発行し、p-op(3) まで打ち切り、より多くの p-ops を発行する等である。ＣＰＵの動作とＤＥＣの機能的挙動を与えてこのシナリオは可能であるかも知れないし可能ではないかも知れないが、主題点は上記タグ発行挙動の直後を、p-ops の発行と打ち切りとの間の関係とすべきことなのである。前節で説明したように、各打ち切りで各機能ユニットは迅速に空となり、正常動作状態に戻り、打ち切りを忘れるべきである。

擬似Ｏｐの引退

各機能ユニットによって p-opsが処理されると、そのユニットの終了パスを介して終了がＤＥＣに通知され、その機能ユニットによる p-op の完了が指示される。これらはバックエンドによって監視され、追跡されて p-opsが引退する時点が制御される。バックエンドがある p-op の引退を何故遅延させるかには特別な内部的理由はあるが、一般的には p-op が引退する時点を支配する２つの発行が存在する。即ち正常環境における適切なＣＰＵ挙動を確保することと、マクロ命令（及び例外処理シーケンス）の適切な打ち切り可能性を確保することである。

最も基本的な p-op は、全ての機能ユニットがその p-op の（一般的には正常な）終了を通知するまでは引退することはできない。ＤＥＣのデコーダがある p-op を発行すると、その p-opの型に関する情報もバックエンドに渡される。これは、そのp-opを処理するであろう、従って終了することが期待される

機能ユニットを含む。この情報に基づいて、完全に終了、即ち完了した後、他の制約を条件として、バックエンドは可能な限り速やかに p-op を引退させることになる。

単一の、そして短い p-op シーケンスマクロ命令の場合、もし何れかの p-ops に障害例外が検出されれば、DECは全命令（即ちその全 p-op ）の打ち切りを取り扱わなければならない。これは、それらの全部が完了（正常な終了で）するまではバックエンドがどの p-ops も引退させないことを要求する。それらが成功裏に完了すると、それらは全て同時に引退することになる。

7 p-ops未満の最大限界に接近している p-op の場合には、命令打ち切りへのこの接近が望ましいものではなくなることに注目されたい。例えば、ある命令が7 p-op シーケンスであるとすれば、7番目の p-op を発行した後、それ以上の p-ops を発行する前の7 p-opsの全部の完全終了を待機しながらDECは実効的に静止する。長さが7 p-opsより長い p-op シーケンスの場合には、適切な命令打ち切りを支援する上で異なるアプローチが絶対的に必要である。

若干の場合には、これはとにかく命令による若干のメモリ書き込みを実際に発生可能にするある組合わせを介して処理できる。若干の場合には、 p-op シーケンスの初めに1またはそれ以上の特別 p-ops を使用して、この検査を行わなければシーケンス内の後の p-ops の1つまで検出されない例外障害を検出する若干の特別検査を行うことも可能または受け入れることができる。本発明は、これらの特別アップフロント検査の間に実 p-op シーケンスの最初の p-op （等）によって行われる検査を加えることであり、これらの早めの p-ops の1つだけが命令打ち切りをもたらすことができ、全ての遅めの p-ops は例外障害のない実行が保証される。

命令打ち切りを支援するこれらのアプローチを用いると、早めの p-ops だけをそれらが全て成功裏に完了するまで未済のままとすればよい。詳述すれば、これらのシーケンスではシーケンスの数多い p-ops の最初だけをこのようにしてDEC（即ちバックエンド）によって処理すればよく、また残余の p-ops はそのように制約されないことを指示している。DEC内部ではこの効果に関する情報は、各 p-op が発行される順度デコーダからバックエンドへ渡される。特別アップフロント p-ops と実シーケンスの最初の p-op との組合せが全ての例外障害を扱うのに充分であるような多くの場合には、早めの p-ops でさえそれらが各々完了すると直ちに引退させることができる。これは、もし特別 p-ops が命令のバックアウトに重大な影響を及ぼさなければ（即ちそれらがプログラマ可視状態を変更しなければ）容認できる。

P-ops の引退に関する最後の一般的な考察は、たとえあるマクロ命令のあるシーケンスの全ての p-ops が完了したとしても、もし早めの p-op が未だに完了していなければ完了済のおそめの p-ops は引退することはできないということである。これは本質的に、 p-ops は順番に引退しなければならないことを見る別の方策である。しかし古い p-op が完了し、引退できるようになると、その p-op 及びこれらの遅めの p-ops の同等は全て同時に引退することになる。

表7はタグ発行及び終了のシーケンスを示す。シーケンス内の4点A、B、C及びDが示され、4つの間隔の境界を限定している。図4の（A）、（B）及び図7の（A）、（B）はそれぞれシーケンス点A－Dにおいて追跡論理160及び打ち切り論理170のレジスタ内に記憶される情報を示す。単一の p-op または p-ops の群は打ち切り群に属するものとして指定される。打ち切り群は、何れかを完了させるために全て完了しなければならない1またはそれ以上の p-ops からなる。換言すれば、もし打ち切り群内の p-ops の1つを打ち切る必要があればその打ち切り群内の全ての p-ops を打ち切る必要がある。

第1間隔中に p-ops(3,4,5) が発行され、p-ops(4,5)は打ち切り群（AG）に属する。図6（A）は打ち切り論理レジスタ内にある追跡中の情報を示す。詳述すれば、p-opが発行されると、p-op情報はタグ番号に対応する位置に記憶され、 p-ops (3,4,5)のための状態レジスタがセットされ、それらの p-ops を発行されたものとして指定する。 p-ops(3,5) のための打ち切りビットがセットされ、打ち切り群 p-op(3)に属する p-ops (4,5) がある打ち切り群の唯一の番号であることを指示する。

第2間隔中に p-op(6)が発行され、 p-op(3)の正常終了が通知される。図6（B）から明白なように、 p-op(6)のための状態ビットが状態レジスタ190（6）内にセットされ、 p-op(3)のためのAP終了ビットが終了レジスタ192（3）内にセットされ、そして正常AP終了が終了記憶レジスタ198（3）内に書き込まれる。

第3間隔中に p-ops(7,8,9) が発行され、p-ops(7,8)はある打ち切り群に属する。この間隔中にIEUは p-op(3)が正常に終了したことを指示し、APは p-op(4)が正常に終了したことを指示し、そして p-op(5)が正常に終了したことを指示する。図7（A）は p-ops(7,8,9) のための状態ビットが状態レジスタ190（7）、190（0）及び190（1）内にセットされ、IEU終了ビットが終了レジスタ192（3）及び190（6）内にセットされ、AP終了ビットが終了レジスタ192（4）内にセットされることを示している。対応する正常終了が終了記憶レジスタ198（3）、198（6）及び198（4）内に書き込まれる。 p-op(3)が引退可能であり、状態レジスタ190（3）内の状態ビットが取り消されていることに注目されたい。

第4間隔中には、未済として許される最大数である7未済 p-opsが存在しているために、付加的な p-ops は発行されない。この間隔中に、APは p-ops(5,6,7) が正常に終了したことを指示し、IEUは p-ops(4,5,9) が正常に終了したことを指示する。しかし次いでAPは p-ops(7) が異常に（例えばページ障害）終了したことを指示し、それに次いでIEUは p-ops (7) が正常に終了下ことを指示する。この結果、p-ops(4,5,6) が引退可能となり、それらは最早未済 p-ops として指示されなくなる。しかし、p-ops(8)が異常終了しているために、p-ops (8) の打ち切り群の一員である p-ops(7) と、 p-ops(8) の後に発行された p-ops(9) も打ち切らなければならない。従って

打ち切り論理１７０は７のＡＴｅｇをタグ状態バス上に発行し、あたかもp-ops[7,8,9]が発行されなかったかのように機能ユニット（この場合ＡＰ及びＩＥＵ）を戻さなければならないことをこれらの機能ユニットに通知する。

擬似Ｏｐ発行制約

バックエンドは未済 p-opsと各機能ユニットの p-op 終了を追跡し、バックエンド内の保持条件論理１６５も未済 p-opsのステータスを使用して付加的な p-opsの発行を制御する。ＣＰＵの正しい総合動作と特定の機能ユニット（特定的にはＤＥＣ、ＡＰ及びＮＰ）内の論理の特定のブロックの動作とを保証するために、バックエンドは種々の型の未済 p-opsの最大数に関する種々の制約を連続的に課す。動作中にこれらの制約によって課される限界に到達すると、バックエンドは保持条件信号をデコーダへ送って次ぎのサイクルに発行されるp-opを遅延させなければならないか否かを制御させる。

バックエンドはほぼ半ダースの保持条件信号を生成してデコーダへ送り次ぎのp-opを潜在的に遅延させる。デコーダはこれらの信号を使用し、現在デコード／アセンブルされているp-op及び通知された保持条件が適用されるか否かに基づいて実際のp-opデコード／発行保持を発生する。各保持条件は１またはそれ以上の（類似）制約に対応する。任意の制約の場合には、バックエンドが未済が最大数であることと、これらのp-ops の１つが将に完全に終了したことを決定すると、対応する保持条件信号が発生する。

多数の制約の場合には、関連する型の最古の未済p-opが完全に終了する最初のp-opであることが保証される。また若干の制約の場合には、保持条件は、単に全ての未済（即ち引退していない）p-ops に基づくのではなく、未済で完全に終了していない p-opsに基づく。ある p-op が完全に終了すると、たとえ更に数サイクルに亙って未済のままとなっても、それは最早特定の機能ユニットのハードウエア制限に伴う若干の制約には無関係である。

バックエンドはデコーダへ保持条件を提示する主発生器の１つではあるが、保持条件の源は他に幾つか存在する。このような保持条件は現 p-op の発行に関して適用されるかも知れないし、適用されないかも知れない制限を通知する。 p-op 発行制御に関して完全に一般的とするために、各クロックサイクル中、擬似Ｏｐパスは有効 p-op かまたは空 p-op （これは多分打ち切り動作と共に）の何れかによって駆動されると言うことができる。デコーダの観点からすれば、以下の何れかが発生しない限りデコーダは常に有効 p-op を発行する。

１）バックエンドからの打ち切り優先、
２）バックエンドからの保持、
３）ＢＰＣからの保持、
４）ＶＩＢ（仮想命令バッファ）からの保持、
５）接頭のみのデコード、
６）２サイクル p-op の第２半分の送り。

勿論、５）及び６）はデコーダが生成するものであり、４）及び５）はマクロ命令シーケンスの第１ p-opsにのみ適用できるものである。

"ＢＰＣからの保持"は、デコーダが次のマクロ命令をデコードしようとしてＢＰＣ内にキャッシュできる「制御の転送」命令（若干の型の転送制御命令はキャッシュされない）を見出した時に発生する。このような命令に対してデコーダは、あるエントリ（対エントリの目標流）の予測情報へＢＰＣアクセスを試みる必要がある。この制御の転送命令に対するＢＰＣアクセスは命令のデコード中に発生する。もしこのＢＰＣアクセスサイクルをデコーダが使用可能でなければ、ＢＰＣ保持が生成される。もし予測情報に関してＢＰＣへのアクセスが使用でき、ミスが発生すれば、たとえＢＰＣ目標流アクセスが使用できなくてもデコーダは進行することができる。もしヒットが発生し、ＢＰＣの関連部分へのアクセスが使用可能でなければＢＰＣ保持が生成される。そうでない場合にはデコーダは予測情報を用いて進行することができ、一方ＢＰＣエントリの目標流はこの転送制御命令に割り当てられた新しい命令待ち行列内へダンプされる。

"ＶＩＢからの保持"は、デコーダが次のマクロ命令をデコードしようとしているが、必要な全命令バイト（命令長に対して）を受信していない場合に発生する。検出済の有効接頭バイトを渡したデコーダは、少なくとも有効演算コードバイトを有するか、またはＶＩＢ保持が強制されなければならない。この演算コードバイトの予備デコードに基づいてもしmod r/m バイトが必要であれば、これも提示されるかまたはＶＩＢ保持が再度強制されなければならない。更に、mod r/m バイトの予備デコードに基づいてもしs-i-b バイトが必要であれば、s-i-b バイトに対しても同じことが適用される。これらのバイトが有効であるとすれば、最終命令バイト（実際にはそれを含むＶＩＢ語）が調べられ（そして暗示的に全ての中間バイトも）、もし有効でなければ（即ち"悪い"または"空"）ＶＩＢ保持が生成される。

"接頭のみのデコード"は、デコーダが次のマクロ命令をデコードしようとしているが、それまでに接頭しかデコードされておらず、現在は更に２つの接頭がデコードされている場合に発生する。１接頭及び第２空バイトの場合は、第２バイトが空でなくなるまで"ＶＩＢからの保持"として取り扱われるか、または１接頭バイトが消費されＶＩＢが前進した"接頭のみのデコード"として取り扱われる。

"第２半分…の送り"は、デコーダが２サイクルの p-op の第１サイクルを将に発行する時に発生する。このサイクル中に特別な空 p-op が付加的な p-op 情報と共に送られ、次のp-opのデコード及び生成は遅延される。

"バックエンドからの保持"は、発行される p-op の型のために、バックエンドの信号に基づいて、デコーダが p-op を直ちに発行するのは"安全"ではないことを理解すると発生する。以下にバックエンドによって強制される全ての未済 p-op 制約を列挙する。

１）７つの合計 p-ops、
２）２つの制御の転送 p-ops、
３）単一のステッピングモード内の１つの打ち切り群、
４）セグメントレジスタ再割り当てを伴う２つの p-ops、

５）ＤＥＣ静止後の０の更なる第１ p-ops。

７つの合計未済 p-opsの最大数は全ての引退していない p-opsに適用される。一般に、従ってこの制約の場合、 p-opsは順番に完全終了しない。しかしバックエンドだけは p-opsを順番に引退させることができる。

最大２つの未済制御の転送 p-opsは、これら全ての p-opsに適用されるが、より正確にはこの制約は実際には制御の転送マクロ命令とそれらの p-op シーケンスの第１ p-opsに適用される。この制約の場合、制御の転送 p-opsは、それらが未済であり且つ完全に終了していない間だけ書きをなす。この p-op が完全に終了したが未だに引退していない場合は、ハードウエア制限に関して最早重要ではない。命令取り込みページ相互要求が生成される時点と、如何にそれらが処理されるかに依存して、たとえ２つの転送制御 p-opsが未済ではなくともバックエンドはこの保持状態を通知することができる。しかし比較的古い順次命令流に関して未済命令取り込み待ち時間が存在する全ての場合には、この制約に対するインパクトは存在しない。ＩＥＵは制御の転送 p-ops（ＩＥＵを含む p-ops）を順番に終了させることが要求されることに注目されたい。

p-op 単一ステッピングが可能になると（ハードウエアのデバッグの目的から）、一時に１打ち切り群の p-opsが発行され、完全に終了され、次ぎの群が発行される前に引退する。

セグメントレジスタのために使用される再割り当てスキームの故に、データセグメントレジスタ（即ちＤＳ、ＥＳ、ＦＳ、ＧＳ）のためのセグメントレジスタ再割り当てを含む２つの未済 p-opsしか存在できない。セグメントレジスタ読み出し専用の、またはＣＳ及びＳＳの両方もしくは何れか一方に記憶される p-opsには、この制約は適用されない。その目的は、 p-ops を記憶する何れかの、及び全てのセグメントレジスタに設される打ち切り可能性を保証することである。ＡＰ静止挙動がＣＳ／ＳＳ記憶 p-opsに既に適用されているので、ＣＳ及びＳＳへの記憶を含む必要はない。

ＤＥＣ静止 p-op が発行されるとデコーダは更なる p-opsを順次発行し続けることができるが、ある更新された制御ビット情報がＡＰからバックエンドによって受信されるまでは次のマクロ命令のデコードを遅延させなければならない。これらの制御ビットは、デコーダのマクロ命令デコード及び p-op アセンブリプロセスに影響するＥＦｌａｇｓの種々のビットである。デコーダが依存するｌまたはそれ以上のＥＦｌａｇｓビットに変更をもたらす p-op は、ＤＥＣ静止 p-op として取り扱わなければならない。これによってこれらのビットのＤＥＣコピーが、更なるマクロ命令のデコードが発生する前に更新されるようになる。予測される更新がＡＰから受信されるまで、バックエンドは保持条件を生成して更なるマクロ命令デコードと第１ p-opの発行とを禁止する。

打ち切り無効を除き全てのデコーダ保持条件は、デコーダが次のデコードサイクルを始動させなければならない時点までに（即ち、現在活動の命令待ち行列を前進させるために制御等を準備し、新たに活動の待ち行列にアクセスして新しいＶＩＢ内容を発生させ、そして予測デコードを遂行するのに遅れないように）デコーダの次の動作状態を決定可能とするのに充分に早めに決定される。デコーダが生成する p-op は破棄され、空 p-op によって置換されるから、打ち切り無効は後刻まで生成されないし、生成する必要もない。同時にバックエンドによってデコーダは生成すべき新しい p-op シーケンスにジャムされ、ベクトル化される。（注：タイミング及びベクトル宛先に対して１以上のジャム及びベクトルの型が存在する。）

上述のように、種々の（ＤＥＣ内部）ユニットによって通知される正常保持条件の場合、デコーダは各機能からの実際の保持信号と対話せず、デコーダはこれらの信号を受信しない。代わりとして各ユニットは保持条件信号を送り、これらの信号は生成中の p-op の型を表す状態信号と組合わされ（論理積され）て実際の保持信号を発生する。これらの信号は、デコーダが生成する付加的な保持信号と組合わされ（論理和され）て総合デコーダ保持信号を発生する。これは p-op 発行及びデコーダ状態シーケンシングを制御するだけではなく、他のユニットにも送られて、デコーダとの対話に頼ってそれらの状態シーケンシングに影響を与える。

機能ユニット静止

機能ユニットが p-opsを処理する際に、それらはプログラマ可視及び関連状態に対する変更を打ち切るか、またはバックアウトする能力を確保しなければならない。これらは、全ての共通して変更した性能限界状態、汎用レジスタ、不動小数点レジスタ、及び殆どのセグメントレジスタ、ＰＣ、及びステータスフラグを含む。他のもの、即ち滅多に変更されることがない特別状態は、履歴スタックを介して、またはレジスタ再割り当てを使用して戻されることはない。代わりにこれらは、支配している（１または複数の）機能ユニットによってこれらが変更できる時点を制限することによって処理される。このプロセスを静止と名付ける。

本質的に、任意の特別なレジスタの場合、（１または複数の）所有者は、関連 p-op が最古の未済 p-op となるまで変更の遂行を遅延させる。このようになると別の（早めの）p-opのためにその p-op が打ち切られる可能性はなくなる。更に、その打ち切りをもたらすであろうこの p-op を原因とする考え得る理由は、多分既に検査済の筈である。従って、今は変更を遂行するのに安全と考えられる。（もし後に支配／変更機能ユニットがその p-op のバックアウトの理由を検出できれば変更を取り消すことができるように、何が必要であってもそのようにしなければならない。）

もしある p-op をＡＰに加えて他の機能ユニットによっても処理されれば、その支配中の機能ユニットだけが異常終了を通知できる。これら全 p-opsは、他の機能ユニットが常に正常終了を通知するように限定され、書き込まれる。もし２つの機能ユニットが共に特別なレジスタを支配していれば、それらは各々それら自身のコピーを変更し、その p-op は両ユニットが常に正常終了を通知するようになろう。

任意の場合には、ある p-op によって変更される特別状態に依存する機能ユニットだけが静止に巻き込まれる。その p-op を処理する他の全ての機能ユニットは正常に挙動する。本質的

に、ある p-op の静止は局所化された基準で、且つ必要な場所だけで発生する。なるべく多くのＣＰＵは正常処理を継続し、（１または複数の）静止機能ユニットによる p-op 処理だけが多分維遅される。

特どの特別レジスタをＡＰが支配している限り、特どの静止 p-ops はＡＰだけによる静止を要求する。これらの多くはＡＰだけの p-opsであり、一方残余はＡＰ／ＩＥＵ p-opsである。ＮＰによる静止（ＡＰ／ＮＰ p-ops上の全部）は、それが処理する３つの制御レジスタに対する変更のためである。デュアル機能ユニット静止の場合は、現在ＡＰ及びＩＥＵに制限されている。これは、ある p-op がＥＦｌａｇレジスタの方向フラグを変更する時に発生する。ＡＰ及びＩＥＵは共に最新コピーを維持しているから、ＡＰ及びＩＥＵは並列ではあるが、独立した静止を遂行する。

たとえ任意の p-op を処理中にある機能ユニットが静止しても、これは必ずしもそのユニットがその p-op の処理を開始する前に権利を静止することを意味しない。特にＡＰが静止の場合には、静止前にその p-op の処理の一部を遂行することができる。ＡＰに必要なことは、特別なレジスタを変更する点において静止することだけである。静止が完了するとＡＰは変更を遂行し、処理を続行することができる。

ＤＥＣも静止を遂行することはできるが、これは他の機能ユニットによって遂行される静止にやや似ているだけである。ＤＥＣ静止 p-op の発行に続いて、ＤＥＣは若干の p-opsのアセンブリ及び発行を遅延させる。この遅延は、ＡＰからの「制御ビット更新」をＤＥＣが受信するまで発生している。ＤＥＣ静止の更なる説明は先行節を参照されたい。「制御ビット更新」に関しては、後述のＡＰ終了パスの節を参照されたい。

ＤＥＣ静止の場合には、ＤＥＣがＡＰから制御ビット更新を受信する他の場合と同様に、若干の特別制御ビットのＤＥＣコピーが更新される。これはＡＰによるこれら制御ビットのそれ自身のコピーの変更と共に発生する。ＤＥＣによって保持されているコピーは、ＤＥＣが所有しているマスタコピーとしては見られないが、その代わりにＡＰによってＤＥＣ内に維持されている二次コピーとして見られる。ＤＥＣはこれらのビットに対する更新をバックアウトする能力は有していない。しかし、ＡＰもこれらのビットのマスタコピーを変更せねばならず、またそれ自身のコピーを変更する前に制御ビット更新を送らないであろうから、これが問題となることはない。これはＡＰ静止を要求し、従ってＤＥＣの制御ビットコピーの更新は関連 p-opが最古の未済となるまでＡＰによって効果的に遅延させられる。

<u>異常終了処理</u>

前述のように、バックエンドは各種類の p-opsの終了を監視し、全ての未決 p-opsに関するステータスを累積する。この情報に基づいて、バックエンドは p-opsの引退（一般的には全ての関連ユニットによって正常終了した後、対異常終了）を制御し、新しい p-opsがデコーダによって発行される時点に影響を及ぼす。 p-opsが完了し、１またはそれ以上の異常終了を受信するとバックエンドは適切な応答を決定する責も負い、次いでそれを適切な時点に開始する。

バックエンドは所与の p-op の異常終了を含む終了を受信すると、一般に全ての予測される終了を受信してしまうまで、それらを累積する。もし異常終了が存在すれば、その p-op は引退することを許されない。この時点で、バックエンドは適切な応答を開始する。もし複数の異常終了が存在すれば、バックエンドは異常終了に対する応答を優先させ、選択する。異常終了処理のこれらの局面を以下に説明する。

応答を開始する前のこの待機は、早めの／古い p-op 異常終了応答（これらは後刻検出され開始される）によって入れ子され／取り替えられる異常終了応答から生ずる対話の場合を処理する設計上の複雑さを最小にするために行われる。また例外処理の開始をもたらす異常終了の場合だけをこのようにして処理するものとすれば、併用によって性能に重大なペナルティが課せられることはない。

バックエンドによって開始される特定応答は、当該異常終了と古い p-opsが未決か否かとに依存する。これは、当該 p-op に明示的に依存するのではなく、特にその p-op の演算コードに明示的に依存しない。応答は適切なタグ（これは必ずしも異常終了 p-op のタグである必要はない）を有する打ち切りサイクルを送ることが多い。打ち切りサイクル中に、または併発打ち切りを伴わない空 p-op を発行するサイクル中に、バックエンドはデコーダがデコード及び p-op が発行した演算を続行する状態へデコーダをジャムし、ベクトル化する。例外処理を開始しなければならない場合には、デコーダはマクロ命令処理へ戻る時にアセンブル及び発行する適切な p-op シーケンスにベクトル化される。開始する例外の型に依存して、異常終了した p-opは打ち切り内に含ませてもよいし、または正常のように引退させてもよい。

応答する異常終了が例外処理をもたらさないような特どの場合には、 p-op が完全に終了すると即否が開始される。僅かな特別異常終了の場合には、その終了がバックエンドによって受信された直後に応答が発生する。これらの終了は正常終了とは考えられないが、それ以上に有益である。これらの終了は、関後の終了が予測されること、及び機能ユニットから特別異常終了の生成を要求されることから真の終了ではない。

これらの場合の対する応答は例外処理の開始を含む上述の応答に類似し、ある適切な p-op シーケンスにベクトル化するだけではなく代わりにマクロ命令へ戻ってベクトル化する可能性を含む。換言すれば、 p-op シーケンス内の進めの p-opsはうちきられ、デコーダはマクロ命令流のデコードを（現命令待ち行列または異なる命令待ち行列からの）次の命令から続行する。また少ない異常終了の場合、応答が直接デコーダに影響を与えないか、ＤＥＣに内部的に他の演算を開始させるかの両方または何れか一方をもたらす。

<u>ＩＥＵ終了パス</u>

表５は、５ビットＩＥＵ終了パス６２のフォーマットを示す。このパスは標準ＣＭＯＳスタイル時分割１／０を使用し、p-ops の正常終了と２つの型の異常終了（例外及び誤予測分岐方向）とを通知する。＃２にパスは３ビット p-op タグと２

ビット終了Ｉｄとを供給する。

ＤＥＣのデコードのタイミング及び p-op アセンブリパイプライン故に、もしＩＥＵ終了コード及び関連 p-op タグがφ2－φ1に（即ち1フェーズ早く）時分割で送られると、ＤＥＣは正しい次の p-op （正しい次のマクロ命令からの、または適切な例外処理 p-op シーケンスからの）が後続する打ち切りサイクルで直ちに応答することができる。

一般に、ＩＥＵは p-opsを順不同で（ＤＥＣによる発行の順番に対して）終了することができ、またそのように終了するであろう。 p-opsを処理／実行する順番に関する限りにおいて同じ型の２つの p-ops間の相対直列化をＩＥＵによって維持しなければならないような若干の p-op の特定の場合が存在する。一般的に、これらの場合の実行順序は決定的であって、終了順ではない。ＩＥＵが条件付き（近）制御の転送 p-opsだけを見る制御の転送 p-opsは、相対直列順に終了することがＩＥＵによって要求される。ＤＥＣの観点からすれば、これらの p-ops を順番に処理することが絶対的に必要ではない。

ＩＥＵが p-opsを処理するに当たって、それらを何時終了できるかに関して２つの場合がある。１） p-opsが実行後にＤＸＢｕｓ転送を要求しない場合、その p-op は正しい終了を知った時に終了することができる。２）実行後に p-opsがこのような転送を要求する場合にはその p-op は、転送が確実に発生することを知るか、または実際に発生していることを知ると終了できる。何れの場合も終了はこれらの時点の後に発生することができる。換言すれば、１）の場合には、もし終了が無条件に正常であればＡＬＵ動作中に、またもし終了がＡＬＵ動作に依存していればＡＬＵ動作が完了した直後に、 p-op を終了させることができる。２）の場合には、ＩＥＵが転送に関してＤＸＢｕｓ調停に勝利したことを知ると、 p-op を終了させることができる。

一般的なＩＥＵパイプラインと、出力待ち行列タイミング及び命令と、ＩＥＵＴｅｒｍ（即ちφ2－φ1）のタイミングとに基づいて、現在ではＩＥＵの以下の実際の終了挙動が予測される。結果をＤＸＢｕｓを介して転送する必要がない p-opsの場合には、終了はＡＬＵ動作サイクル中に開始される。殆どの p-ops の場合、これは無条件に正常終了であり、転送制御 p-opsの場合の正しい終了はＡＬＵサイクルの最初の部分中に決定される（これは INTO 命令 p-op にも適用される）。時には、終了パス上へ出て行くことができないこの終了は待機させられ、後刻（しかし、勿論極めて直ちに）ＤＥＣへ通知される。

結果をＤＸＢｕｓ上へ転送する必要がある p-opsの場合には、終了は転送サイクル中に開始される。この場合も、もし終了が直ちに出て行くことができなければ待機させられ、後刻送られる。

上記１）の場合に属し、異常終了をもたらし得る、そしてＡＬＵ動作に依存するＢＯＵＮＤ及びＲＥＰｅｄ列マクロ命令に関連する p-opsの場合には、１）の場合の上記タイミングは作業しない。これらの場合 p-opsは、あたかもそれらが結果をＤＸＢｕｓ上へ送出する必要があるかのように取り扱われる。

終了が何故順不同に生成されるかについては２つの理由がある。第１に、ＩＥＵは処理／実行する p-opsを順不同に選択する。第２に、実行順序に対してＩＥＵは p-opsを更なる順不同で終了することができる。換述すればＩＥＵは、１） p-opsの場合は直ちに終了し、２） p-opsの場合には先ずＤＸＢｕｓ上へ進まなければならない（多分そのようにするにはＩＥＵのデータ出力待ち行列内で待機してから）。後者の場合、これらのp-ops は、それらが実際にＤＸＢｕｓ上へ進むと終了する。付加的に、待ち合わせが１）及び２）終了を（一時的に）超えるものとすれば、待ち合わせした終了に先立って若干の高い優先順位終了（例えば制御の転送終了）が通知される可能性もある。（勿論制御の転送 p-opsの相対直列化は確保しなければならない。）

制御の転送終了には関係なく、全ての場合にＩＥＵは p-op を終了する前にその処理を完了させなければならない。これはＡＰからＩＥＵへのレジスタ更新をもたらすか、または単純にメモリオペランドをレジスタへ転送する p-opsを含む。両型の p-ops に関して、その p-op が終了する前に発信オペランドを受信しなければならない。これをＡＰの挙動と対比して説明すると、種々の転送およびレジスタ更新の場合、ＡＰは何が効力のあるレジスタ更新であるかを受信する前に終了することができる（組合わせるレジスタ結果を必要とするかも知れないとしても）。

ＩＥＵは p-op を処理中に検出した異常に応答して異常終了を通知した後は、その p-op が正常に終了したかのように他の p-ops の処理を続行する。ＩＥＵは p-opsの処理を停止せず、ある意味では異常終了に対する来たるべき応答を待機する。

ＩＥＵ終了

以下に表５に示す終了を説明する。

知らせるべき実終了が存在しない場合には終了を通知してはならない。終了パスは全クロックサイクル中有効であり、常に何かを指示しなければならない。

正常終了は、ある p-op の処理中に異常が検出されなければ通知される。

誤予測分岐方向終了は、予測した分岐方向が正しくない場合に制御の転送 p-ops（これらは条件付き近制御転送でなければならない）上で通知される。これは、正しく予測された分岐方向の場合の正常終了の代わりである。

異常終了は例外理由のためのものであり、対応するアーキテクチャ的に定義された例外を通知するためにそれぞれ使用される。除算誤差はＤＩＶ及びＩＤＩＶマクロ命令の p-op シーケンス内のＥＵａｂｏｒｔと共に注釈をつける p-ops上に使用される。境界検査及びＩＮＴＯ溢れはそれぞれＢＯＵＮＤ及びＩＮＴＯ命令のＥＵａｂｏｒｔ p-ops上で使用される。ＲＥＰ' ｅｄ命令繰り返し停止終了はＲＥＰ' ｅｄ列マクロ命令の p-op シーケンスの p-ops、即ちＥＵａｂｏｒｔと共に注釈をつけるp-ops 上に通知される。もしその p-op によって遂行される試験が、列マクロ命令の繰り返しを停止させるべきことを指示すれば、この終了は正常終了の代わりに通知される。もし p-op 試験が列命令の繰り返しをを始動させるべきではな

い（即ち0繰り返しの連行）ことを指示したとしても、これが適用される。これらの状況において例外が検出されなければ、正常終了が通知される。

1つの p-op に多重異常をIEUが検出する可能性はなく、従ってIEU異常終了間に相対優先順位は発行されないが、他の機能ユニットの終了に対して優先順位が発行される。IEU内の所与の p-op 型に対して1つの型の例外しか存在し得ないから、DEC打ち切り論理はその p-op に基づいて例外の型を独特に識別できる。IEU異常終了は、AP及びNP異常終了に対してDECによって認識されたそれらの優先順位に基づいて幾つかの群にグループ化される。殆どの異常終了は中間優先順位群内にグループ化され、一方REP停止終了は低優先順位を有する。

誤予測分岐方向終了は、全てのAP終了に対して固定された特定の優先順位を有していない点が特別である。その代わりにAP終了と組合わされた実行分岐方向（予測した方向及び予測の正しさ）がDECバックエンドによって開始される動作を決定する。

AP終了パス

表6はAP終了パス60のフォーマットを示す。このパスは標準CMOSスタイル時分割I／Oを使用し、p-ops の正常終了及び種々の異常終了を通知する。

DECデコードのタイミングとp-opアセンブリパイプライン故に、もしAP終了コードが1フェーズ早めに（φ2－φ1）時分割で送られれば、DECは正しい次の p-op （次のマクロ命令からの、または適切な例外処理もしくは他の p-op シーケンスからの）が後続する打ち切りサイクルで直ちに応答することができる。終了コードの符号化は、重要な場合に、DECは理想的な応答時間を提供できるようになっており、別のp-opを発行するかもしくは打ち切り、または正しい次の p-op を発行する。他の例外の場合は、応答時間内に効果的な特別サイクルが存在する。即ち打ち切りサイクルの前に1つのサイクルが発生し、次のサイクルに正しい次の p-op が後続する。

殆どの異常終了を処理するこの特別サイクルはDECのバックエンド間に配分されて何が起こったのか及びどうするのかが見出され、またDECのデコーダがジャムされてベクトル化され、正しい次の p-op のデコードが開始される。迅速終了の場合バックエンドは制限された処理状況を有する。この迅速処理を支援するのは、APが常に p-opsを順番に終了させるので、バックエンドが次の終了に関連付けられるp-opタグを予測できることである。

迅速終了は、あるp-opの正常終了、及び任意選択的に誤予測アドレス及びDビットの両方または何れか一方をも指示する制御ビット更新（APからDECへ）のような状況に対して投げられる。正常終了の場合、p-opタグ及び支配下にある p-op の型に関する情報を有するバックエンドは、保持条件信号内のこの終了をデコーダと分岐制御論理とに反映させる必要がある。誤予測アドレス及びDビットの両方または何れか一方を伴わない制御ビット更新の場合、終了パス転送は当該制御ビットのための更新値を供給し、その後にデコーダはマクロ命令流のデコードを続行することが可能になる。もし誤予測アドレス及びDビットの両法または何れか一方も指示されていれば、この終了のタイミングは実効的に他の全ての非迅速終了と同じようになる。

前述のように、APは p-opsを順番に（DECが発行する p-opsの順番に対して）終了しなければならない。これはAPが p-ops を処理する順番とは無関係であるが、他の理由からAPが p-opsを処理できる順番に対して制約が存在する。全ての場合、 p-op はそれが完了した後は何時でも終了させることができる。しかし、IEUの状況とやや類似して、 p-opsを終了させることができる最も早い時点に関して2つの場合が存在する。場合1は p-opsが実行後にDXBus転送を要求しない場合であって、正しい終了を知ると p-op を終了させることができる。 p-opsがこのような転送を要求する場合2では、転送が確実に発生することを知ると p-op を終了させることができる。換言すれば、場合1では全てのシステムメモリの（異常終了に対する）参照及び必要検査が完了すると p-op を終了させることができる。場合2では、転送に関してAPが、DXBusまたはPAdrBus同停に勝利したこと及び転送が確実に発生するであろうことを知ると p-op を終了させることができる。これはPAdrBusメモリアドレス参照転送がTLBミスのために打ち切られる場合を含み、この終了は転送が実際に完了するか否かを知る前には発生することができない。若干の終了に特定して、DECによる付加的な制約／要求を以下に説明する。

IEU、NP、またはメモリからの汎用レジスタ更新の受信を除き、更新が受信される前にAPは処理済 p-op を終了させることができることに注目されたい。更新は本質的にそれ以上の処理を要求することはなく、単に適切なレジスタ内へ記憶させ、これを表すためにレジスタインタロック制御を更新するだけでよい。APは、関連 p-op が完全に終了する時点までに、従ってそれが引退する前にこれらの更新を受信することを保証されている。勿論、それでもAPは打ち切り発生に関して予測されるレジスタ更新を適切に追跡していなければならない。

APは、ある p-op の処理中に検出した異常に応答して異常終了を通知した後に、その p-op の処理を適切に終らせる。終了に依存してAPは更なる p-op の処理を中止することができる。この挙動は、DECが例外処理を開始して応答するような異常終了の後に発生する。他の全ての場合、APは処理を続行する。

処理を中止した後、APは必要内部状態の退避及び凍結の両方または何れか一方を行い、異常終了に対する来たるべき応答を待機する。この応答は発生しないかも知れず、より一般的にはAPは例外処理を開始する全ての応答に関知しなければならない。

AP終了

以下に表6に示す終了を説明する。 p-ops上に例外を指示する全ての異常終了に対して、終了Idのビット<3..0>が処理を開始すべき例外の割り込み数に直接的に対応することに注目されたい。これに対する2つの例外は、特別な場合に使用され

る代替「デバッグ及び一般的保護」障害コード（即ち1111 010 X）である。「遮断異常終了」（コード＝1111 1001）も、例外処理が発生せず代わりにＤＥＣが遮断されることが特別である。

通知すべき実終了が存在しない場合には終了を通知してはならない。ある p-op の処理中に異常が検出されない場合には正常終了が通知される。

「制御ビット更新」は、全てのＤＥＣ静止 p-opsと共に使用される。これらはＩＦ、Ｄ、及びＢビット（ＥＦｌａｇｓ及び種々のセグメント記述子内に見られる）の全部または何れかに直接または間接的に影響を与える p-opsである。ＡＰが影響を受ける（１または複数の）ビットの（１または複数の）新しい値を決定すると、この終了は更新値をＤＥＣへ送るために使用される。

これは真の終了ではなく、特に制御ビット変化をもたらすp-opを終了させることがないことに注目されたい。通常のp-op終了が未だに要求され、制御ビット更新の後に発生しなければならない。（制御ビット更新自体は先行p-opの終了に後続しなければならない。）ＤＥＣは、制御ビット更新終了を受信した後ならば何時でも、p-opの終了には無関係にp-op発行を続行するであろうことにも注目されたい。概念は、p-opの処理中に、影響を受けた制御ビットの新しい値を知ると直ちにＡＰはＤＥＣにこの更新を送り、その p-op の処理を続行するということである。

上記制御ビットはプログラマ可視ビットを表すから、ＡＰ及びＤＥＣは潜在的にこれらのビットに対する変更をバックアウトできなければならない。これを（性能に重大な衝撃を与えることなく）回避するために、ＡＰは制御ビット更新を通知する時に（後になってではなく）これらのビットのマスタコピーを変更し、当該 p-op が最古の未決 p-op となるまでこれらの両動作を遂延させる。本質的に、制御ビット更新を通知することは、更新を通知する前にＡＰ静止を暗示する。

制御ビット更新の第２の形状は第１の形状に類似しているが、更新の転送をバス上の「誤予測アドレス」及びＤビットの両方または何れか一方にも指示する。これは、ＤＥＣが目標アドレスを予測した（及びＤビットが変化しないものとした）制御p-ops の転送に使用される。もし制御 p-op を転送するためにＤＥＣが予測した（物理）目標アドレスが正しくなければ（即ちＡＰが発生した（物理）アドレスと異なれば）、ＡＰはこれを通知してＤビットの更新値を送らなければならない。勿論、ＡＰはアドレス更新（即ち、正しい目標アドレス）も命令キャッシュタグへ送らなければならない。

ＡＰは、ＰＡｄｒＢｕｓを通して正しい目標アドレスを送出し、同時に誤予測アドレス及びＤビットの両方及び何れか一方と共に制御ビット更新を通知する（同時実行が要求される）ことによってこれをすべて行う。この更新は、更新制御ビット値を送ることに関しては上記第１の形状に類似する。更に、ＤＥＣは誤予測を表すようにある内部状態を適切に変更し、正しいアドレス及びＤビットを用いて命令取り込み及びデコードを再開する。前述のようにＤＥＣは、ＤＥＣが次の有効マクロ命令をデコードできるようになる前に更新された制御ビットを受信することが本質的に保証されている。

第１の制御ビット更新とは異なりこれは真の終了であり、特定的にはその転送制御 p-op を終了する。ＡＰが正しい目標アドレスを送出し更新を通知するように与えられているタイミングは、もし別の異常（即ち例外）検出されれば「制御ビット更新終了」の通知を回避することを可能にする。即ち、ＡＰはアドレスを送出し更新終了を通知するか、または異常終了を（無効アドレスと共に）通知するかの何れかである。

異常をもたらすページクロスの場合、ＰＡｄｒＢｕｓ転送は発生しない。これは、セグメントオーバーラン（一般的保護障害をもたらす）、ページ障害の何れかまたは両方の故であり得る。ＡＰは異常終了を通知して障害が発生したことを表示する。もし命令実行がページ境界を交差することを真に必要とすれば、例外処理が開始される。ＡＰから見れば、ページクロス要求の処理及び終了は周囲の p-opsとは無関係である。ＤＥＣはp-opの流れ及び p-op 例外に対してページクロスに例外を適切に優先させることを重要視する。

例外のための各異常終了は対応するアーキテクチャ的に定義された例外を通知する。２つの場合（例えば「一般的保護」障害）には、例外を通知する１対の終了Ｉｄが存在する。一方は一般的に使用され、他方は他の機能ユニット（すなわちＩＥＵ及びＮＰ）による異常終了に対して異なる優先順位を有する限り区別する必要がある若干の例外環境において使用される。

これらの異常終了の若干が特定のマクロ命令に関係付けられていることに注目すべきである。特定的には、「３８７使用不能」、「無効演算コード」、および「一般的保護」（コード＝1111 0100）終了は関連 p-op シーケンスの最初の p-ops上で通知される。「一般的保護」終了（コード＝1111 0100）及び「デバッグ」終了（デバッグ障害に対してコード＝1111 0101）はマクロ命令シーケンスの最初の p-ops上で通知される。「デバッグ」終了（デバッグトラップに対してコード＝1111 0001）はマクロ命令及びタスクスイッチシーケンスの最後の p-ops上で通知される。

ＭＣＣ終了バス

ＭＣＣ　２５終了バス６５は、標準ＣＭＯＳスタイル時分割Ｉ／Ｏを使用する１ビットバスである。実際の信号転送はφ１－φ２境界で発生し（即ちＭＣＣ終了はφ２転送である）、他のフェーズ境界での転送は定義されていない。このバスは p-opsから直接生じた正常メモリ書き込みの終了を通知するために使用される。メモリ読み出し、システムメモリ参照、及び他の参照（例えばＩ／Ｏ）のための終了は生成されない。

ＭＣＣはメモリ参照アドレスを順番に（メモリ参照をもたらす p-opsの発行順に対して）ＡＰから受信する。ＭＣＣは、この順番にメモリ書き込み参照を終了させなければならない。このため、終了を通知するのに p-op タグの明示転送は必要ではない。書き込みの順番終了に基づいて終了バスを監視しているＤＥＣのバックエンドは、どの p-op タグがＭＣＣからの次の終了に組合わされるかを予測する。

メモリ書き込みの終了は、アドレスをＡＰから受信しそれを

適切な書き込み予約待ち行列内に配置する時に通知される。これは、ＭＣＣがそのデータを受信する時及び書き込みが待ち行列を出る時には無関係である。 p-op による読み出し・変更・書き込み操作の書き込みも終了する。誤った位置合わせをもたらす、または４バイトメモリ書き込みより大きい p-opsの場合には、ＡＰが１より多くの隣位置合わせされたアドレスを生成する必要がある。このような p-op の書き込みの終了は、最後のアドレスが予約待ち行列内に配置されると通知される。

メモリ書き込みをもたらす p-opsのＭＣＣ終了には拘りなくＡＰはそれ自信のこれらの p-opsの終了を生成する。これは、ＡＰが１またはそれ以上の隣位置合わせされたアドレスの最後のアドレスをＰＡｄｒＢｕｓを介してＭＣＣに転送する時に発生する。ＭＣＣは通常（１または複数の）アドレスを直ちに待ち行列内に配置することができるから、通常はＭＣＣがメモリ書き込みアドレスの受信を指示する必要はない。しかし（待ち行列が一杯であるか、または待ち行列の１つの中の先行（古い）書き込みと重なり合うために）ＭＣＣがアドレスを適切な書き込み予約待ち行列内に配置できないような場合には、ＭＣＣによる終了が必要である。これらの後者の場合には、ＤＥＣがp-op発行を進めるのを防ぐために終了が遅延される。

もしＭＣＣが、遅延させることができるそれ自信の終了を有していなければ、次のことが発生し得る。ＡＰが p-op を終了すると、ＤＥＣは書き込みを生成した p-op が完了し書き込み予約待ち行列内で安全であると確信する。ＤＥＣはこの書き込みアドレスに対応するタグを過ぎた７またはそれ異常の p-op の p-op タグを適切に追跡し続ける。

ＡＰが p-op を正常終了させるが対応するメモリ書き込みを生成しない特別な状況も存在する。これらの場合にはＡＰは「正常終了、しかし書き込みなし」を通知して書き込みが発行されないこと、従ってＭＣＣからの終了が期待されないことをＤＥＣに指示する。

ＮＰ終了バス

要約すれば、ＮＰ終了は２ビットのバスであり（ p-op を順番に終了させることを前提としている）、浮動小数点突き合わせ例外を通知する。ＣＰＵ内には任意選択ＮＰを含むための論理が設けられているが、詳細説明は省略する。

レジスタ再割り当て

前述のように、命令を流出させるのに必要な事象までＣＰＵの状態を戻すために使用されるメカニズムの１つはレジスタ再割り当てである。この技術は必然的に、プログラマ可視の（即ち、仮想）レジスタの集合をより大きい集合の物理レジスタ内に写像することを伴う。物理レジスタの数は、少なくとも未決とすることが許され且つレジスタを変更することができるp-opの最大数だけ仮想レジスタの数を超えている。この技術は汎用レジスタファイル及びセグメントレジスタファイルの両者に適用される。

特定のマクロ命令アーキテクチャ（80386)は、ＶＲ０～ＶＲ７と名付けた８つの仮想汎用レジスタと、６つの仮想セグメントレジスタとを提供する。前述のように、多くとも合計７つのp-ops と、セグメントレジスタを変化させる多くとも２つのp-opsを未決とすることが許される。これと調和して、ＡＰ １５は、ＦＲ１～ＦＲ１５と名付けた１５の物理汎用レジスタの集合と、８つの物理セグメントレジスタとを含み、一方ＩＥＵ１７は１５の物理汎用レジスタを含む。物理レジスタＰＲ０がＩＥＵ内に存在しているが、これは他の目的のために使用される。

図５は仮想レジスタＶＲ０～ＶＲ７から物理レジスタＰＲ１～ＰＲ１５への写像の概要図である。各物理レジスタは"Ｖ"で概略的に示す対応付けられた有効ビットを有する。これらの有効ビットは機能ユニットによって以下のように使用される。汎用レジスタ再割り当てを支援するために、バックエンドレジスタ再割り当て論理１７５はポインタ集合アレイ１７７と自由リストアレイ１７８とを維持している。ポインタ集合アレイ及び自由リストアレイは各々８リストの記憶を提供し、これらの各リストは未決 p-opsのタグの最下位の３ビットに対応する３ビット索引を有している。各ポインタ集合及び各自由リストは図中に列（colusn）によって表されている。

所与の索引のためのポインタ集合及び自由リストは、その索引に対応するタグを有する p-op の発行の直前の状態を維持する。ポインタ集合は仮想レジスタＶＲ～ＶＲ７に対応する８つのエントリを含み、各エントリは物理レジスタの１つを指すポインタを含む。自由リストは、ポインタ集合の一員（メンバー）によって指されていない物理レジスタを指すポインタを含む７つのエントリを含む。

タグ＝０を有する p-op の発行前の初期状態を考えよう。こ

の初期状態では、ＶＲ０がＰＲ８に、ＶＲ１がＰＲ７に、ＶＲ２がＰＲ６に、…、そしてＶＲ７がＰＲ１に写像される。また自由リストはＰＲ９乃至ＰＲ１５を指すポインタを含み、ＰＲ９がリストの先頭でＰＲ１５がリストの尾である。この状態はタグ＝０が先頭のエントリの列内に記憶される。

さて、以下のタグ０、１及び２を有する３つの p-opsの代表シリーズを考えよう。

タグ＝０：　ＶＲ０＝ＶＲ０＋ＶＲ３

タグ＝１：　ＶＲ３＝ＶＲ３＋ＶＲ５

タグ＝２：　ＶＲ４＝ＶＲ０＋ＶＲ３

ＶＲ０は先にＰＲ８上に写像されているから、p-op(0) は、p-op(0) が完了可能となることが確立されるまでＰＲ８を変更することはできない。従って p-op(0)の始動前に存在する写像は、ＶＲ０を自由リスト内の物理レジスタ上へ写像するように変更しなければならない。ＰＲ９が自由リストの先頭であるので、ＶＲ０がＰＲ９上へ写像される。ＰＲ８は、８つの p-opsが発行され p-op(0)が引退したことが保証されるまでは先頭に立たないから、自由リストの尾に配置される。自由リスト内の他の各項目は先頭に向かって前進する。従ってタグ＝０と共に発行される実際の p-op はＰＲ９＝ＰＲ８＋ＰＲ５である。

次の p-op 、即ち p-op(1)はＶＲ３を変更しようとする。この p-op のバックアウトを可能ならしめるために、ＶＲ３は自由リストの先頭にある物理レジスタ、即ちＰＲ１０上へ写像される。ＰＲ５は自由リストの尾に配置され、ＰＲ１１は自由リストの先頭に前進する。タグ＝１と共に発行される実際の p-op はＰＲ１０＝ＰＲ５＋ＰＲ３である。

p-op(2)はＶＲ４を変更しようとする。従ってＶＲ４は物理レジスタＰＲ１１に写像され、ＶＲ５が自由リストの尾に配置される。タグ＝２と共に発行される実際の p-op はＰＲ１１＝ＰＲ９＋ＰＲ１０である。

物理レジスタを変更する p-op がある機能ユニットに到着するとそのレジスタの有効ビットはクリアされ（無効を表し）、その p-op が終了した時にのみセットされる（有効を表す）。これは、物理レジスタを読み出そうとしている過めの p-op のために正しいデータが存在していることを保証するために必要である。図示の特定例においては、p-op(0)がＰＲ９を変更し、p-op(1)がＰＲ１０を変更する。p-op(2)はＰＲ９及びＰＲ１０の内容を要求するからそれが実行可能となる前に有効源レジスタ（ＰＲ９及びＰＲ１０）を有していなければならない。これは p-op(1)及び p-op(2)が終了した場合にのみ発生する。p-op(0) 及び p-op(1)は、もし何れかが一掃されるとp-op(2) も一掃されてしまうから、引退してはならないことに注意されたい。

<u>データキャッシュサブシステム内の書き込み待ち行列</u>

図６は、データキャッシュサブシステムの制御を提供するＭＣＣ　２５のブロック線図である。そのジョブは、ＡＰ　１５が発生しＰＡｄｒＢｕｓ　５５を介して引き渡される書き込みアドレスを、幾つかのチップの何れかによって発生されＤＸＢｕｓ　５８を介して引き渡される対応データに結び付けることと、書き込みデータ（３２ビット倍長語内に右寄せされている）とＡＰによって指定されているバイトアドレスとをバイト位置合わせすることと、書き込みと関後の読み出しとの間の同一アドレスに対するメモリデータ従属を検査し、データが使用可能になると直ちにそれらを短絡することと、書き込み動作を発生させた p-opsが成功裏に終了することが保証されるまで書き込み操作を待機させることによって実行の緊密さ（コヒーレンス）を維持することと、必要な場合には主メモリまたはキャッシュ自体を変更することなく書き込み操作を打ち切り可能ならしめることである。

データキャッシュスブシステムは３つのカテゴリのデータ操作を扱う。正常データアクセスは、ＮＰ　２０が進行するもの（もしあれば）を除いて、プログラマ指定のデータアクセスである。他の２つのカテゴリはシステムアクセス及びＮＰアクセスである。各カテゴリ内のメモリから読み出されるデータは何れかのカテゴリの早めの p-opsによってなされた書き込みを表していなければならないが、異なるカテゴリの書き込みは非同期的に処理することができる。即ち異なるカテゴリの近くの（実行順序に対して）書き込みは同一アドレスを変更しないこと、またはもしそれらが行えばカテゴリ間の書き込みの非同期性の効果は温和であるものとしている。

ＭＣＣ　２５は、書き込みバッファ３０２及びマルチプレクサ３０３と組合わされた書き込み予約待ち行列（WRESQ）３００と、システムバッファ３０７と組合わされたシステム書き込み待ち行列（SYSWQ）３０５と、ＮＰバッファ３１２及びマルチプレクサ３１３と組合わされた書き込み予約待ち行列（NPWQ）３１０とを含む複数の待ち行列構造を含む。

ＷＲＥＳＱ　３００は正常データアクセスのみに役立つ。これは、各書き込みデータ（これは単一バイト、１６ビット語、または３２ビット倍長語であってよいが、実行ユニットからの単一の３２ビット倍長語内に常に右寄せされて到着する）を、任意バイト境界上でメモリ内の位置合わせを指定できる（１または複数の）対応アドレスによって指示されるように位置合わせすることと、何れかのカテゴリの書き込みと関後の読み出しとの間の同一アドレスに対するメモリデータ従属を検査することとを含む前述の全ての機能を遂行する。

ＳＹＳＷＱ　３０５は、システム書き込みを発生させた p-opsが成功裏に終了し、それらがメモリ内へ書き込まれるまでシステム書き込みを提供する。これは多くとも４つの未済システム書き込みを提供する。システムアクセスは隠システム構造（ページ終書エントリ、ページ表エントリ、セグメント記述子、及びタスク状態セグメントデータ）にアクセスするためにＡＰによって進行されるアクセスである。全てのシステム書き込みは「アクセスされた」または「話中」ビットをセットする単一の倍長語読み出し・変更・書き込み操作として発生する。ＡＰは順不同の実行を進行しないから、全てのシステムアクセスは順番に発生する。さらに、システム書き込みは読み出し・変更・書き込み操作から発生するので、アドレスは書き込みデータの前にＭＣＣに到着しなければならない。

ＮＰＷＱ　３１０は８つのＮＰ書き込みアドレス（少なくとも２つのＮＰ p-opsの結果を保持するのに充分）を提供する。

NPに指令され、NPから指令されるNPデータアクセスは3つの主な点が正常データアクセスとは異なる。即ち、単一のNP p-op は10バイトまでのデータを読み出し及び書き込みの両方または何れか一方を行うことができるが、正常 p-op は多くとも4バイトのデータにしかアクセスできない。従ってNPは、単一の p-op によって指定された書き込み操作を遂行するために多重倍長語転送を遂行することができる。NP p-op のためのデータは常にMCCに順次に（即ちアドレスが到着するのと同じシーケンスで）到着する。

WRESQ 300は最も複雑な書き込み待ち行列であり、後述するように p-op 終了及び打ち切りの処理を行う。WRESQは8つのエントリを受け入れる複雑なデータ及び命令バッファからなる。各エントリは、倍長語アドレス（倍長語は32ビットのデータ）のための30ビット幅の連想記憶装置（CAM）レジスタ、数値比較論理と「最終」ビット及び「解放された」ビットとを含む4ビットの専用タグCAM、及び各データバイトのための有効ビットと全データレジスタのための「現行」ビットとを含む制御論理と組合わされた4バイト幅のデータレジスタを含む。

WRESQは、データ物理アドレスバス待ち行列（PAdrQ）320と呼ばれるFIFOバッファからデータアクセスのためのメモリアドレスを受信する（これらのメモリアドレスはAPから到着するとPAdrQに蓄積される）。各アドレスは、遂行されるアクセスの型（読み出し、書き込み、または読み出し・変更・書き込み）と、アドレスを発生した p-op のタグと、アドレスされた倍長語へ及び／またはそれから転送される倍長語のバイトを指示する4ビットのバイト可能マスクと、そのアドレスがその p-op によって発生される最後のものであるか否かを指示する「最終」ビットとを伴う。

書き込みまたは読み出し・変更・書き込みアクセスのためにPAdrQから受信したアドレスは、アドレスを伴うバイト可能ビットによって指示される何れかのバイト位置内に有効ビット集合を有するWRESQ内に既に入力されている全てのアドレスと連想的に比較される。もしWRESQ内に既に重なり合った書き込みを指示する何かを見出すと、WRESQ内への新しいアドレスの処理は、重なり合った書き込みがメモリへ書き込まれWRESQから除かれるまで見合わさなければならない。

この場合MCCは、位置がメモリへ書き込まれるまで中断して、書き込み待ち行列により多くのアドレスを受け入れるようにしなければならない。これはパイプライン機能停止と呼ばれ、この場合MCCはアドレスをPAdrQ内に戻すことを可能とし、もしこの猶予が過れの前兆を示せば、MCCはPAdrBusをロックしてAPがより多くのアドレスを発行するのを阻止する。そうでなく、もしパイプライン機能停止が要求されないか、または重なり合いエントリを除去することによってこのような機能停止が解決された後は、新しいアドレスがWRESQ内の位置に割り当てられる。

WRESQ 310内の位置は割り当てカウンタによるラウンドロビン方式の割り当てのために選択される。もし選択された位置が自由であればアドレスは「アドレスCAM」内にコピーされ、タグ及び「最終」ビットはTag CAM内にコピーされ、4つの「現行」ビット及び「解放された」ビットは0にセットされ、4つの「有効」ビットは書き込まれる倍長語のバイトを指定するバイト可能ビットに対応してセットされる。一方もしWRESQ位置が再割り当てのために取り上げられた時に未だに使用中であれば（1またはそれ以上の有効ビットがその位置においてセットされていることにより指示される）、その位置がメモリに書き込まれるまでMCCはそれ以上のアドレスの受け入れを中断（パイプラインを機能停止）させなければならない。

新しいエントリがWRESQ内に書き込まれるクロック期間に、またはその後に、有効ビットがセットされているデータバイト内にデータが書き込まれる。実行ユニットが、書き込まれるデータを提供する前にAPがアドレスを送信する保証はなく、またMCC自体がアドレスが到着し次第それらを処理できるという保証もない。従って、データはWRESQエントリが確立される前に既にMCCへ送られているかも知れない。Bエントリ WBuf 302がこれを受け入れる。このWBufはDXBus（書き込みデータをMCCへ引き渡すバス）とWRESQ自身の入力との間に配置されている。DXBusに到着するデータは、それが表している操作の型（もしWRESQに宛てられていれば正常メモリ書き込み）及びそれが発生したp-opのタグによって識別される。

正常メモリ書き込みデータがDXBusに到着すると、それはその4ビット p-op タグの最下位3ビットによってアドレスされた32ビットWBuf内に記憶され、この p-op タグの最上位ビットはエントリと共に記憶され、（後述するようなTag CAMヒットが発生しない限り）「現行」ビットはエントリのためにセットされる。同時に、そのタグはWRESQのTag CAM内で探索される。もしそのデータのための位置（または2つの隣接位置）が、「最終」ビットがセットされている1つの位置を含むWRESQ内に見出されれば、そのデータは直ちにその（またはそれらの）位置に（この場合WBufエントリの「現行」ビットがセットされていない）書き込まれる。同様に、「最終」ビットがセットされているアドレスがWRESQ内に入力されると、そのアドレスを発生した p-op のタグに対応するWBufエントリが質問され、もしその「現行」ビットがセットされていればデータは全てWBufエントリへコピーされ、WRESQ「現行」ビットはセットされ、WBuf「現行」ビットはクリアされる。

上述の2つのメカニズムによって、データまたはアドレスのどちらが先に到着するかには関係なく、またはそれらが同時に到着してもデータ及びアドレスの両者が存在する場合にはアドレス及びデータは共にWRESQ内へ入力され、その p-op のためのWBufエントリの「現行」ビットはクリアされ、（1または複数の）WRESQエントリの（1または複数の）「現行」ビットはセットされる。この時点でWBuf位置は再使用のために自由となる。データはアドレスに対して順不同で到着し得るから、できる限り早く処理を発生できるようにするため

に、データレジスタ及びＷＲＥＳＱの「現行」ビットへの２つの独立した経路が設けられている。ＷＢｕｆから発する一方はその位置内に書き込むことができ、この位置には（ラウンドロビンカウンタによって選択された）対応アドレスが同時に書き込まれる。ＤＸＢｕｓインタフェースから直接の他方はＴａｇＣＡＭによって識別される（１または複数の）位置内に書き込むことができる。これにより新たに到着するアドレスとＷＢｕｆからのデータとを対にして、先に確立されているＷＲＥＳＱエントリ内にＤＸＢｕｓから新たに到着したデータが書き込まれるクロックサイクルと同じクロックサイクルに、新しいエントリ内に書き込むことができる。

ＷＲＥＳＱ内に入力されるデータは、それがメモリ内で占めるであろうバイト位置と同じバイト位置内にそれをバイト位置合わせするローテータを通過する。ＷＲＥＳＱ内への２つのデータ経路毎に別個のローテータが設けられている。（多分２つの隣接エントリの）（第１）ＷＲＥＳＱエントリの最下位バイト位置から数えて０の値を有する隣接「有効」ビットの数は、ＷＲＥＳＱ内へのデータ書き込みが発生する前に位置合わせのためにデータを回転させなければならない左方へのバイト位置の数を示している。「有効」ビットに組合わされている論理は、もし先行ＷＲＥＳＱ位置も同じ p-op タグのためのアドレスを含まなければそしてその場合に限って、ある位置の「有効」ビットをゲートすることによってこのデータをバレルシフタへ供給する。

データがＷＲＥＳＱ内のある位置に書き込まれると、それはこの保証が得られるのである。

アドレスは、ＰＡｄｒＱから抽出されると各々先にＷＲＥＳＱ内に（及び他の２つの書き込み待ち行列内にも）入力されている全てのアドレスと（書き込みアドレスに関して説明したようにして）連想的に比較される。前述のように、到来書き込みアドレスと現存ＷＲＥＳＱエントリとの重なり合いは、早いエントリがメモリへ書き込まれて書き込み待ち行列から除かれるまでパイプライン機能停止をもたらす。しかし、たとえ同一倍長語（の異なる部分）を変更したとしても、重なり合わない書き込みは待ち行列内に入力することができる。読み出し操作及び読み出し・変更・書き込み操作（アドレス読み出し）のためのアドレスも書き込み待ち行列エントリと連想的に比較される。書き込みと同様に、この比較は読み出しアドレスのバイト可能ビットと待ち行列エントリの対応「有効」ビットとの論理積によって決定されるバイト毎に並行される。

もしが読み出しアドレスによって指定されたバイトをアドレスするＷＲＥＳＱエントリがなければ、またはもし読み出しアドレスによって指定されたバイトをアドレスする各エントリ（書き込み・待ち行列ヒット）の「現行」ビットがセットされていれば、ＭＣＣはＤＣＩ　３７へ通知してそのアドレスの正常キャッシュ探索を進行させる。（どのキャッシュアクセスもキャッシュミスの場合の遅延、及び要求されたデータを検索するための主メモリ操作の必要をもたらす。）

一方、もし読み出しアドレスが、「現行」ビットがセットされていない１またはそれ以上の書き込み・待ち行列エントリ内（もしＴａｇ　ＣＡＭによってアドレスされていれば）同じタグ値を有する任意隣接位置か、または（もし新エントリ割り当てカウンタによってアドレスされていれば）以前のエントリ割り当ての方向に隣接し「最終」ビットが無効にされた位置にも書き込まれる。書き込まれるデータは多くとも４バイト幅であるから、１つの倍長語のためのバイト位置と突き合わせるためにセータをバイト規模で回転させ、次いで隣倍長語を書き込むことは、メモリ内の倍長語境界にまたがる位置合わせされていない書き込み操作のために４バイト全てを隣倍長語内の適切な位置に同時に書き込むことになる。

「最終」ビットがセットされた正常カテゴリアドレスがＰＡｄｒＱから抽出されると、ＭＣＣはＭＣＣ終了信号をＤＥＣへ供給する。これらのアドレスは順番に（即ちＤＥＣから発行されるそれらを生成した p-opsと同じ順番に）処理され、ＤＥＣはどの p-opsが正常メモリアクセスを生成するかを知っており、たとえＭＣＣ終了が（１または複数の）アドレスが処理された p-op のタグを明示的に含まなくとも、ＤＥＣはそのＭＣＣ終了をその p-op に明確に対応付けることができる。ＭＣＣからの終了によってＤＥＣは、ＷＲＥＳＱエントリが未だに確立されていない全ての p-opsからのデータを受け入れるためには最悪の場合には８より多いＷＢｕｆが必要であること、及び打ち切りの場合には無関係なデータ及びアドレスを待ち行列から適切に流出させ得ることを保証できるようになる。ＤＥＣは、正常アクセスを発生し未だにＭＣＣによって終了されていない最古の p-op の他に７より多い p-opsを発行しないから、

でヒットすれば、ＰＡｄｒＱからのアドレスの処理はこれらの全てのエントリのためのデータを受信するまで中断（パイプライン機能停止）させなければならない。機能停止が解決され、キャッシュデータが使用可能になると、ＭＣＣはＤＣＩに指令して書き込み待ち行列ヒットがセットされた「有効」ビットを有していないバイトだけをＤＩＯＢｕｓ　５７上へゲートさせる。アドレスをヒットしている全ての書き込み・待ち行列エントリの「有効」ビットによって選択される他のバイトは書き込み待ち行列へ、及びＭＣＣによってＤＩＯＢｕｓ上へ駆動される。従って、未だにメモリへ送られていない書き込みデータは、後の読み出しへ“短絡”させることができる。書き込みが書き込み待ち行列内で未済になっているバイトのために第２の書き込みを受信するとパイプラインは機能停止させられるから、データの所与のバイトをアドレスする１より多いエントリは存在することはできないが、同じ倍長語の異なるバイトを供給する幾つかのエントリは存在できる。書き込み待ち行列はこれらの全てからの「有効」バイトを組合わせてデータを選択し、ＤＩＯＢｕｓ上へ駆動する。

ＣＰＵの他のユニットと同様に、ＭＣＣはタグステータスバスを介してＤＥＣから供給されるタグステータスを追跡しなければならない。各クロックサイクル中にＤＥＣは、最古の未決 p-op タグ（ＯＯＴａｇ）または打ち切りタグ（ＡＴａｇ）の助言を得て２つのメッセージ型の１つをタグステータスバス上へ送信する。ＷＲＥＳＱは「最古のエントリポインタ」（ＯＥＰ）と呼ぶその最古のエントリを指すポインタを維持し

ている。あるエントリは、それがOOTagより古くなるまでメモリへの書き込みに対して無資格のままである。OOTagを受信するるサイクルにOOTagは、1またはそれ以上の「有効」ビットがセットされ「解放された」ビットはセットされていない各書き込み待ち行列エントリのタグCAM内容と比較される。タグ比較は、4ビットの2の補数演算を使用して、エントリの4ビットタグから4ビットOOTagを減ずることによって遂行される。タグは2進計数シーケンス（0000,0001,0010, …,1110,1111,0000,…）で発行され、一時に7より多くないタグが未決であるから、OOTagの値は1つのサイクルから次のサイクルまで多くとも（もし7つの未決 p-opsが全て引退し、新しい p-op が同じサイクルに発行されれば）8までジャンプすることができる。従ってもしエントリのタグからOOTagを減じて得られた差の最上位ビットの値が"1"であれば、それはOOTagよりも8またはそれ以上若くはなり得ないから、エントリのタグはOOTagより1乃至 p-ops古いことを示している。このようにしてOOTagより若いことが見出されるるエントリ各に、そのエントリの「解放された」ビットがセットされる。エントリは、OEPによって指し示されているエントリの「解放された」ビットがセットされ、その「現行」ビットがセットされ、そして1またはそれ以上の「有効」ビットがセットされている場合に、そしてその場合にだけ、キャッシュ及び主メモリの両者または何れか一方へ書き込むことができる。書き込みが発生すると、エントリの「有効」ビットはクリアされ、OEPは1またはそれ以上の「有効」ビット（もし存在すれば）がセットされている次の順次エントリへ前進させられる。

DECが打ち切りを通知すると、PAdrQ、WRESQ及び他の2つの書き込み待ち行列を含む全ての待ち行列内のp-opタグフィールドに対してATagが検査される。この検査はエントリが解放され得る時点を決定する検査と同じようにして、即ち待ち行列内に指定されているタグフィールドからATagを減ずることによって遂行される。もし待ち行列エントリのタグフィールドがATagより大きい（古い）ならばエントリは待ち行列内に維持され、そうでなければその（1または複数の）「有効」ビットはクリアされる。待ち行列の制御論理の実施態様に依存してポインタも調整しなければならないかも知れない。WRESQの場合、もしエントリが削除され、割り当てポインタが最も早く削除されたエントリまで戻って移動し、そしてこれがOEPを過ぎて移動していれば、OEPは割り当てポインタに先行するエントリまで移動する。

WRESQに組合わされているWBufのエントリ及びタグ値によってアドレスされる他の類似構造に対しても同じような検査が行われるが、WBuf内のエントリのアドレスは単にそのタグの下位3ビットであり、エントリのタグの最上位ビット（MSB）だけがそのエントリ自身の中に記憶されているから、ATagの下位3ビットより大きいかまたは等しい3ビットアドレスを有しまたATagのMSBに等しい記憶されたMSBを有するか、またはATagの下位3ビットより小さいアドレスを有しまたATagのMSBに相対するMSBを有する全てのエントリの「有効」ビットをリセットするだけで充分である。

CPUの全ての機能ユニットと同様に、MCCは打ち切りサイクル中に内部バス上に提示されるデータは無視し、打ち切りの後に未だ適切であれば送っていたデータを再送信する。従って、ある単一のサイクル中に、MCC（及びCPUの残余）は未だに発行されたことがないATagより大きいかまたは等しいタグを担持する p-op が有していた状態にそれ自信をリセットする。

IEUにおける疑似Opの処理

図7はIEU 17のブロック線図である。IEUは2つのデータ経路、即ち単一サイクルデータ経路400及び多重サイクルデータ経路405を実現する。単一サイクルデータ経路は、加算、減算、及びぴけた送りのような1サイクル内に完了させることができる全ての整数命令を実行する。多重サイクルデータ経路は、乗算、除算、及びASCII及び10進数演算機構のような複数のサイクルを必要とする全ての整数p-opsを実行する。2つのデータ経路は、レジスタ再割り当てに関して説明したようにして仮想レジスタが写像される物理レジスタを含む共通レジスタファイル410を使用する。

各データ経路は共通バス集合412に結合された要素を含み、バス結合器415が2つのデータ経路の間を分離している。単一サイクルデータ経路は、汎用ALU420と、バレルシフタ422と、符号伝播、先行0及び1方向、等々のための特別論理425とを含む。多重サイクルデータ経路は、乗除算回路430（8×32乗算器アレイ）と、ASCII及び10進数調整のための回路435とを含む。

入力 p-opsは p-op バス52から受信され、 p-op 待ち行列450へ導かれる。マルチプレクサ452は待ち行列内の実行される p-op を選択し、実行される p-op は単一サイクル制御論理455（PLAによって実現）へ通信される。単一サイクル p-op の場合、制御論理455は単一サイクルデータ経路要素を制御する。多重サイクルデータ経路 p-op の場合、制御論理455は p-op の第1サイクルで多重サイクル要素を制御しマイクロコードROM460へアドレスを供給する。マイクロコードROM460は多重サイクル制御論理462（PLA）と共に p-op の最後のサイクルの制御を提供する。

ALU p-opsの場合には、結果はレジスタ内に記憶され、終了は直ちに終了待ち行列470内へ入力され、終了待ち行列470の内容はIEU終了バス上へ送り出される。メモリ書き込みの場合には、結果が直接DXBusへ進められる（この場合終了は終了待ち行列内へ入力される）か、または出力が最後の出力のためのDXBus出力待ち行列475内に配置されるかの何れかである。バスが使用可能になると、終了は終了待ち行列内に入力される。

P-op 待ち行列450の深さは8である。P-op待ち行列は、複数の読み出しポートと1つの書き込みポートとを有している。待ち行列制御論理480は待ち行列を制御して通常はFIFO（先入れ先出し）のように機能させるが、順不同読み出しをも支援する。待ち行列制御論理は待ち行列がエントリを有してい

るか否かを指示する。待ち行列制御論理は待ち行列内の p-cp の位置も識別する。

もし p-cp 待ち行列が、待ち行列が空の時に p-op を受信すれば、その p-op は直ちにデコードされて適切な制御信号が生成される。 p-cp のデコードが進行中の時点に実行準備検査が行われる。この検査はデータオペランド及びフラグオペランド従属、及び順番実行及び機能ユニット直列化のような若干の特別な実行基準を含む。もし p-op が実行準備検査に失敗すれば若干のまたは全ての制御信号は使用禁止となる。もし p-op が実行されなければ、その p-op は待ち行列内に配置される。

もし待ち行列内にエントリが存在すれば、待ち行列はＦＩＦＯの如く機能する。待ち行列の先頭の p-op 、及び待ち行列内の次に若い p-op が読み出される。実行準備論理４８２は両 p-opsに対して検査を行う。待ち行列の先頭の p-op に対する実行準備検査はデータオペランド従属を含む。もし待ち行列の先頭の p-op が実行準備検査に合格すれば、その p-op はデコードされ実行される。もしその p-op を実行することができなければ、それは次の動作サイクルにおける検査のために再発行される。

待ち行列内の次に若い p-op に対する実行準備検査は、データオペランド及びフラグ従属、待ち行列の先頭の p-op に対するインタロック、及びその p-op が（順番実行のような）特別な実行基準の主体か否かを含む。例えば、 p-op が要求する源レジスタ内に有効ビットがセットされているかどうかが検査される。もし待ち行列の先頭の p-op が実行に失敗すれば、待ち行列内の次に若い p-op が実行準備検査の全てに合格していれば、この p-op がデコードされ実行される。もし待ち行列の先頭の p-op 及び待ち行列内の次に若い p-op を共に成功裏に実行することができれば、待ち行列の先頭が実行される。

複数の読み出しポインタ及び１つの書き込みポインタは待ち行列の動作を追跡し続ける。もし次に若い p-op が実行されれば対応する実ポインタは待ち行列内の次のエントリを指すように更新される。もし待ち行列の先頭の p-op が実行されれば、第１読み出しポインタは第２読み出しポインタの値を入手し、第２読み出しポインタは待ち行列内の次のエントリを指すように更新される。書き込みポインタは待ち行列内の第１空位置を指すために使用される。打ち切りサイクル中は全てのポインタは打切りタグと比較され、その結果に基づいて適切な値にセットされる。

待ち行列制御論理４８０は待ち行列内の各エントリ毎にステータスビットを有する。ステータスビットは、新しい p-op を待ち行列内にロードしている間に "有効" にセットされる。もし打ち切りサイクル中に p-op 待ち行列内のエントリを一掃するのであれば、適切なステータスビットが "無効" にセットされる。実行のために識別された p-op がデコードされる。もし実行のために識別された p-op が単一サイクル p-op であれば、単一サイクルデータ経路４００（レジスタファイル、ＡＬＵ、バレルシフタ、及び特別論理）のための制御信号が制御論理４５５によって生成される。単一サイクル p-op は単一のクロックサイクル中に実行される。この時間中には多重サイクルデータ経路４０５は何らの機能も遂行しない。

もし実行のために識別された p-op が多重サイクルp-opであれば、第１状態制御信号が単一サイクル制御論理によって生成される。単一サイクル制御論理はマイクロコードＲＯＭ４６０も賦活する。残余の状態のための制御信号はマイクロコードＲＯＭ及び多重サイクル制御論理４６２から生成される。多重サイクルデータ経路４０５はこの時間中に演算を遂行する。多重サイクル動作は単一サイクルデータ経路からレジスタファイル４１０だけを使用する。

P-opsの同時（並列）実行を遂行することが可能である。もし実行のために識別された p-op が多重サイクル p-op であれば、考え得る性能利益は、待ち行列から次の単一サイクルp-opを実行することによって得られる。単一サイクルデータ経路を使用して単一サイクル p-opsをまた多重サイクルデータ経路を使用して多重サイクル p-opsを実行することができる。多重サイクル p-op に対するデータまたはステータスフラグ従属が存在すれば単一サイクルp-opは実行されない。多重サイクルp-opと単一サイクル p-op との間に資源対立が存在する時間中（レジスタファイルへの書き込み及びステータスフラグ更新中）も、単一サイクル p-op は実行されない。

多重サイクル制御論理は動作の状態を識別する状態機械を有する。整数実行ユニットは４つの状態、即ち単一サイクル、多重サイクル、同時、または遊びの１つを取ることができる。

単一サイクルデータ経路と多重サイクルデータ経路との間のバスは、同時動作中にバス結合器４１５によって切り離される。これらのバスは多重サイクル動作中通常は接続されていて、データファイルからのデータ転送及び（次の p-op のために、その前の） p-op からの結果の使用の両方または何れか一方を可能にしている。

もしある p-op が実行可能であると識別されると、そのp-opは単一サイクル制御論理及び多重サイクル制御論理の両方または何れか一方に指示される。機能ユニットが話中であることを見出せば、 p-op は実行されない。これは p-op 待ち行列制御へ戻して通知され、論理を実行する準備を整える。多重読み出しポインタに対して適切な調節が施される。

通常、 p-op 待ち行列、待ち行列制御論理、及び実行準備論理は、データオペランドインタロックの解決及び特別実行基準に基づいて p-opsの発行を維持しようと試みる。ＩＥＵ内の各種機能ユニット（ＡＬＵ、バレルシフタ、特別論理、乗／除算回路）の制御論理はハードウエア資源対立を解決し、単一サイクル、多重サイクル、または同時の何れかの演算を遂行する。もしＱＮＥＸＴと呼ぶ信号によって資源対立が通知され、発行された p-op を実行することができなければ、 p-op 待ち行列制御論理によって再発行することが要求される。フラグスタック４８５を使用してフラグが追跡される。

結論

以上に本発明の好ましい実施例を完全に説明したが、種々の変更、代替、及び等価を使用しても差し支えない。例えば、上述の実施例は各機能ユニット毎に分離したチップを用いて実現されているが、分散したパイプライン制御を用いる基本アーキ

テクチャは単一チップ実施例においても同じように効果的且つ有用であろう。同様に、この特定の実施例は特定の命令を実行するが、他の実施例が他の命令集合を実行するように設計可能である。

また、タグを機能ユニットへ通信する特定のメカニズム（符号化されたタグを使用するOOTagまたはATagを用いたタグステータスバス）を説明したが、他の可能性も存在する。一時に多くともn p-opsを未決にすることができるシステムにおける1つの可能性は、タグをNビットベクトル内の単一集合ビットとして表すことである（但し、Nはnより大きいかもしくは等しい）。未決 p-opsの収集がNビットベクトル内の集合ビットの論理（論理的なセンスで）群によって表されるようにこれらのタグを順次に発行する。このベクトルは機能ユニットに通信されてステータスが表明され、一方類似の型のベクトルによって打ち切りを通知する。

従って上記の説明及び添付図面は、請求の範囲によって限定本発明の範囲を制限するものではない。

表1　相似Opパスフォーマット

第1φ1に

ビット(単/複)	フィールド
<51..48>	SegReg
<47..45>	
<44..41>	SrcAReg
<40..37>	IndexReg
<36..33>	EASpec
<32>	ASize
<31>	TwoCyc
<30..29>	MemRef
<28..25>	SrcBReg
<24..21>	DestReg
<20>	RegStore
<19..17>	OperSize
<16..14>	OperSpec
<13..4>	Opcode
<3..0>	PopTag

第1φ2に

ビット(単/複)	フィールド
<51..48>	DestSegReg
<47>	LastPop
<46>	(保留)
<45>	Lock
<44..40>	StatMod
<39..32>	Imm8
<31..16>	ImmDispHi
<15..0>	ImmDispLo

第2φ1に

ビット(単/複)	フィールド
<47..14>	(未定義)
<13..4>	Opcode
<3..0>	(未定義)

第2φ2に

ビット(単/複)	フィールド
<47..32>	(未定義)
<31..16>	ImmHi
<15..0>	ImmLo

表2　物理アドレスパスフォーマット

第1φ1に

ビット(単/複)	フィールド
<25>	DTAGReq
<24>	ITAGReq
<23>	DecReq
<22>	MCCRLd
<21>	ARReq

第1φ2に

ビット(単/複)	フィールド
<25..23>	Stream
<22..20>	Operation
<3..0>	InstrNum(*P-op Tag except for Steam 0)

第2φ1に

ビット(単/複)	フィールド
<20>	Lok
<19>	Trm
<18..4>	物理アドレス<16..2>
<3..0>	バイト選択

第2φ2に

ビット(単/複)	フィールド
<19>	Val
<18..4>	物理アドレス<31..17>

表3　DIOBusフォーマット

DIOCtl

φ1に

ビット(単/複)	フィールド
<4>	最終オペランド
<3..0>	フレーム

φ2に

ビット(単/複)	フィールド
<4>	RdData 有効
<3..0>	P-op タグ

DIOBus

φ1に

ビット(単/複)	フィールド
<31..0>	WrData<31..0>

φ2に

ビット(単/複)	フィールド
<31..0>	RdDATA<31..0>

表４　データ交換バスフォーマット

サイクル１　φ２

ビット	フィールド
<21>	APReq
<20>	NPHLd
<19>	NPReq

サイクル２　φ１

ビット(単／複)	フィールド
<21..20>	TT(転送型)
<19..16>	P-op タグ
<15..0>	データ<15..0>

サイクル２　φ２

ビット(単／複)	フィールド
<18..16>	MemOp
<15..0>	データ<31..16>

表５　１ＥＵ終了バスフォーマット

φ２に

ビット(単／複)	フィールド
<4..2>	疑似Ｏｐタグ
<1..0>	終了Ｉｄ

＊疑似Ｏｐタグは、終了するｐ－ｏｐのｐ－ｏｐタグの３つの最下位ビットを含む。

終了Ｉｄ

値	意味
００	終了なし
０１	正常終了
１０	誤予測分岐方向終了
１１	異常終了

表６　ＡＰ終了バスフォーマット

φ２に

ビット(単／複)	フィールド
<3>	終了Ｉｄ、ビット＜７＞
<2>	Ｉｄ＜６＞
<1>	制御ビットＢ／Ｉｄ＜５＞
<0>	制御ビットＤ／Ｉｄ＜４＞

φ２に（通常）

ビット(単／複)	フィールド
<3>	制御ビットＩ／Ｉｄ＜３＞
<2>	制御ビットＮ／Ｉｄ＜２＞
<1>	制御ビットＨ／Ｉｄ＜１＞
<0>	制御ビットＳ／Ｉｄ＜０＞

（次頁に続く）

（前頁からの続き）

終了Ｉｄ　＜７．．０＞

値	意味
00XX XXXX	終了なし
01BD INHS	制御ビット更新
10BD INHS	誤予測アドレス／制御ビット更新
110X XXXX	正常終了
1110 0001	デバッグ
1111 0010	ハイパコード
1111 0100	一般的保護（命令感度）
1111 0101	デバッグ（区切り点）
1111 0110	無効演算コード
1111 0111	３８７使用不能
1111 1000	二重障害
1111 1001	遮断
1111 1010	無効ＴＳＳ
1111 1011	セグメント不在
1111 1100	スタック障害
1111 1101	一般的保護（命令を除く）
1111 1110	ページ障害

表7　擬似Op終了及び途時のシーケンスの例

		APIF	IEUIF
クク	擬似Op	クク 値	クク 値
3	CHK I AG		
4	XFE I		
5	XFE I AG		
A			
6	DEC I AG	3 OK	
B			
7	XFE I		3 OK
8	XFE I AG	4 OK	
9	DEC		6 OK
C			
		5 OK	4 OK
			5 OK
		6 OK	9 OK
		7 OK	
		8 ページ障害	
			7 OK
D			

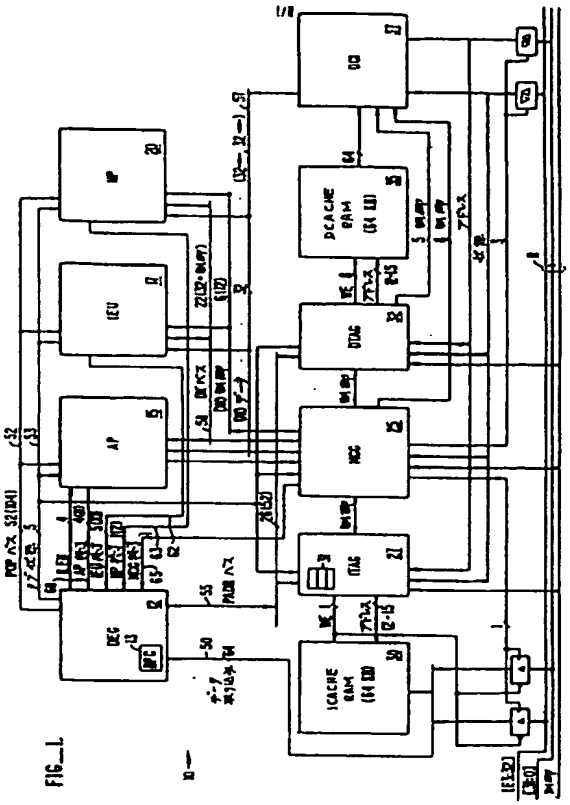


FIG_1

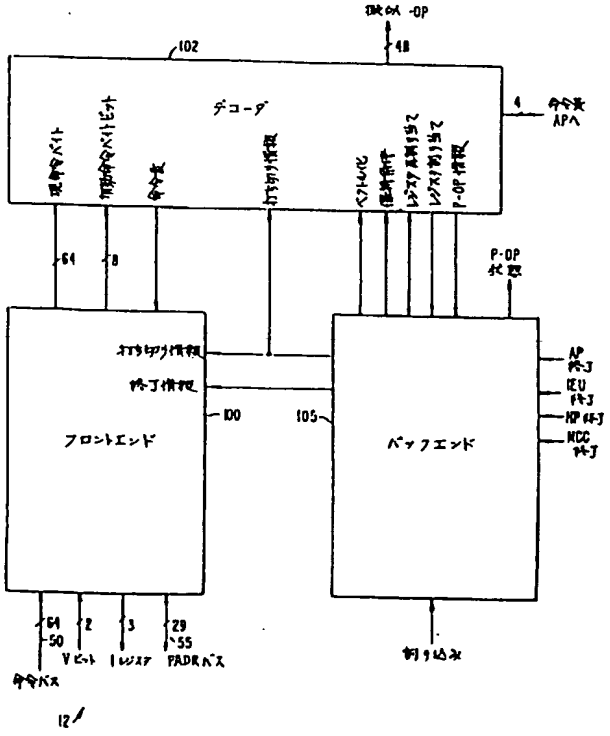


FIG_2

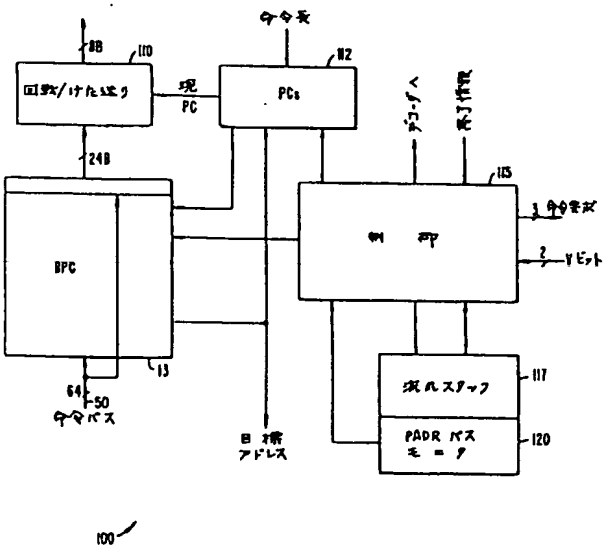


FIG_3A.

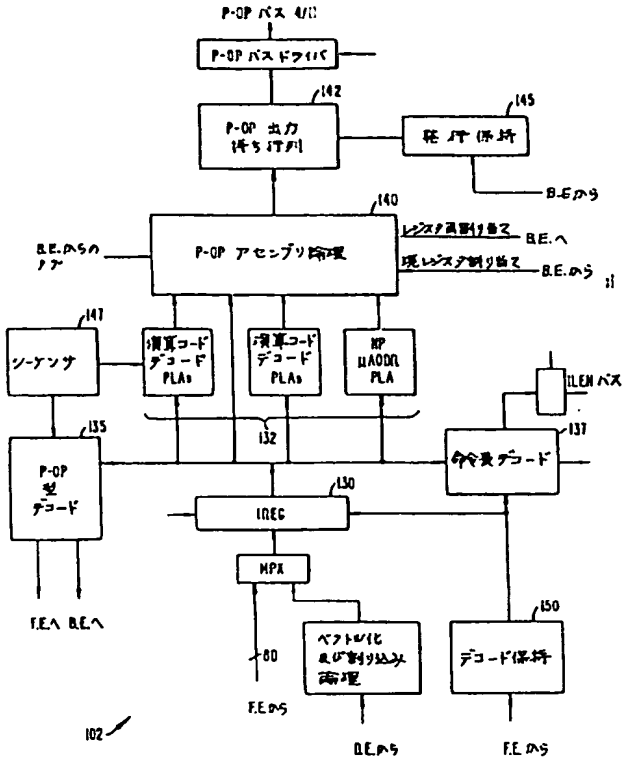


FIG._3B.

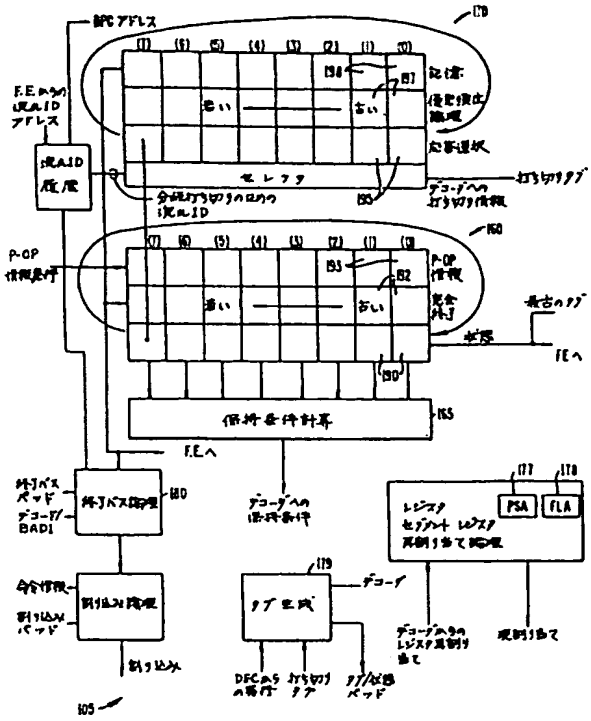


FIG._3C.

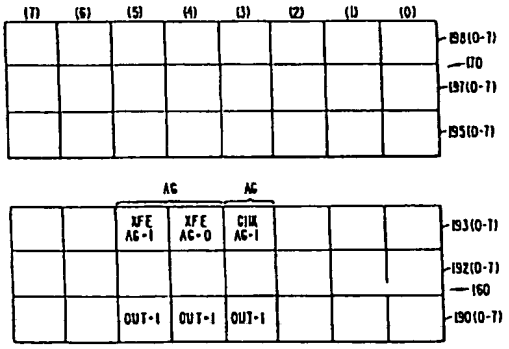


FIG._4A.

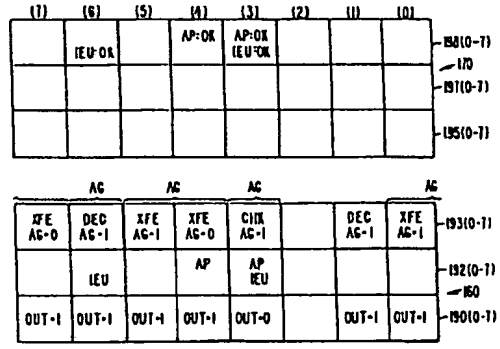


FIG._4C.

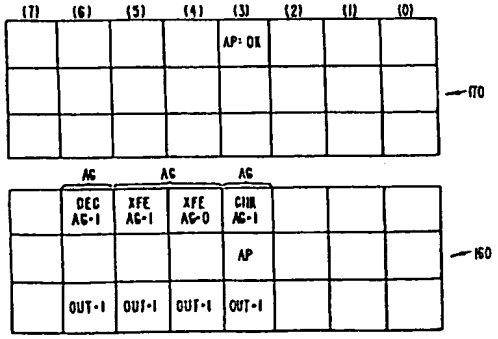


FIG._4B.

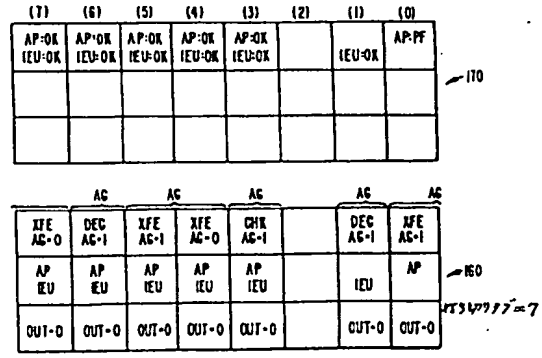


FIG._4D.

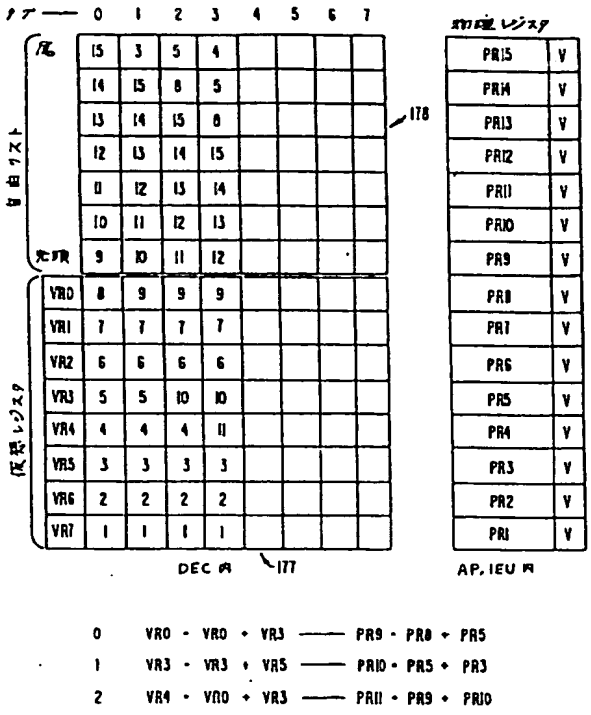


FIG_5.

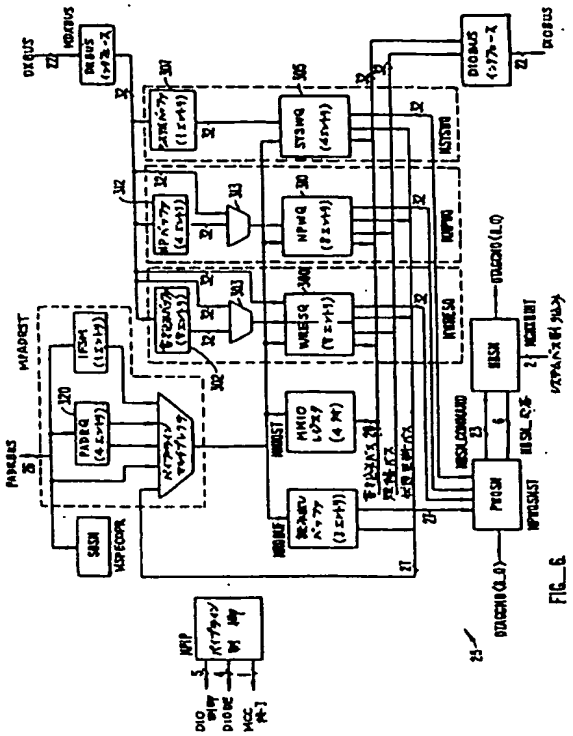


FIG_6

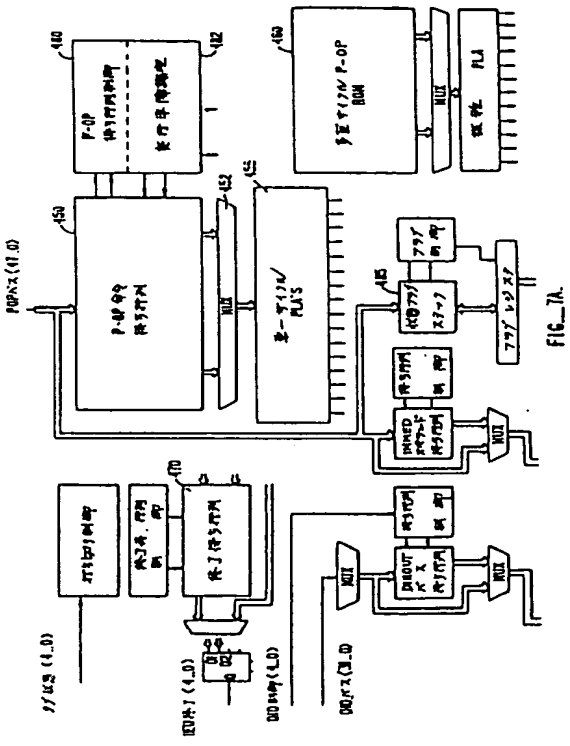


FIG_7A

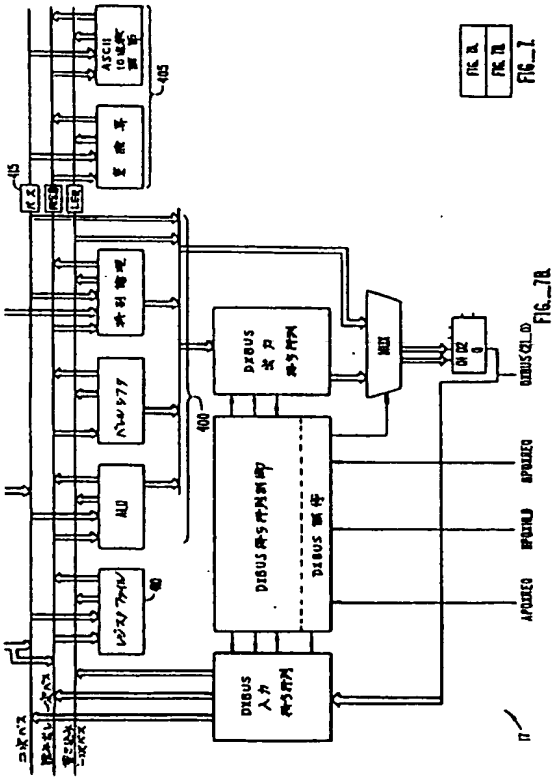
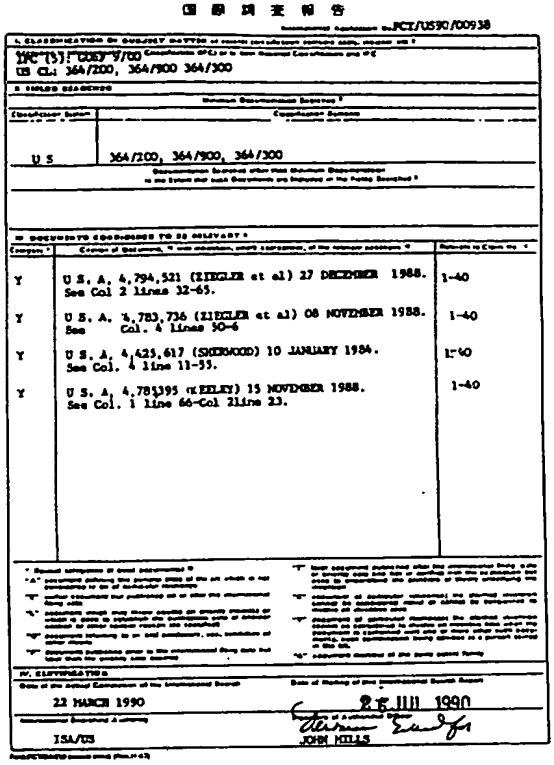


FIG_7B

国際調査報告



Y	U S, A, 4,794,521 (ZIEGLER et al) 27 DECEMBER 1988. See Col 2 lines 32-65.	1-40
Y	U S, A, 4,783,736 (ZIEGLER et al) 08 NOVEMBER 1988. See Col. 4 lines 50-6	1-40
Y	U S, A, 4,425,617 (SHERWOOD) 10 JANUARY 1984. See Col. 4 line 11-55.	1-40
Y	U S, A, 4,783,395 (KELLEY) 15 NOVEMBER 1988. See Col. 1 line 66-Col 2line 23.	1-40

US Cl: 364/200, 364/900 364/300

U S 364/200, 364/900, 364/300

22 MARCH 1990

ISA/US

JOHN MILLS

第1頁の続き

⑦発明者	ステイレス デイヴイド アール	アメリカ合衆国 カリフオルニア 94087 サニーヴエイル ダートシヤー ウエイ 830
⑦発明者	ヴアン ダイク コルビン エス	アメリカ合衆国 カリフオルニア 94539 フレモント カユガ コート 45770
⑦発明者	メータ シユレニク	アメリカ合衆国 カリフオルニア 95148 サン ホセ マウント イザベル コート 3164
⑦発明者	フエイヴアー ジョン グレゴリー	アメリカ合衆国 カリフオルニア 95124 サン ホセ レーサアン コート 5246
⑦発明者	グリーンレイ デイル アール	アメリカ合衆国 カリフオルニア 95124 サン ホセ レツヂウツド ドライヴ 1744
⑦発明者	カルグノニ ロバート エー	アメリカ合衆国 カリフオルニア 94087 サニーヴエイル ムール レーン 1594

【公報種別】特許法第１７条第１項及び特許法第１７条の２の規定による補正の掲載
【部門区分】第６部門第３区分
【発行日】平成９年（１９９７）８月１２日

【公表番号】特表平４－５０３５８２
【公表日】平成４年（１９９２）６月２５日
【年通号数】
【出願番号】特願平２－５０４３８９
【国際特許分類第６版】
G06F 9/38 370
15/16 370
【ＦＩ】
G06F 9/38 370 A 9462-5B
15/16 370 Z 8837-5L

手続補正書

平成9年2月10日

特許庁長官 荒井 寿光 殿

1．事件の表示
特許出願番号 平成2年特許願第504389号
国際出願番号 PCT/US90/00936

2．補正をする者
事件との関係 特許出願人
住所 アメリカ合衆国 カリフォルニア 95131
サン ホセ ノース ファースト ストリート 2202
名称 ネクスジェン マイクロシステムズ

3．代理人
住所 東京都港区赤坂3丁目4番3号
三田第一長岡ビル 電話03－3452－0441
氏名 (7782) 弁理士 鈴木 弘男

4．補正命令の日付 （自発）

5．補正の対象
請求の範囲の欄

6．補正の内容
請求の範囲を別紙のとおりに補正する。

【請求の範囲】

1．発行されるとそれぞれ1つの未決演算の状態を達成する一連の演算を発行する装置と、
未決演算の少なくとも若干をそれぞれが実行できる複数の機能ユニットと、
割り当てられたタグを検査することによって2つの未決演算の相対年齢を決定できるように順序を付けたタグの集合の一員であるタグを各未決演算に割り当てる装置と、
前号の未決演算が完了する時点を決定する装置と、
未決タグの独特さを保証するように未決演算の数を制限する装置と
を具備することを特徴とするコンピュータプロセッサ。

2．上記制限する装置が、一時に多くともn個の演算を未決可能とすることを許容し、
上記タグが2nより大きいか等しいある範囲において順次に発行され、
2つの未決演算の相対年齢をそれらのタグの符号付き比較によって決定できるようにした
請求項1に記載のコンピュータプロセッサ。

3．最古の未決演算を失効する装置と、
未決演算と引退させる演算との間の境界をマークするタグを準備することによって、演算の引退が成功したことを上記機能ユニットの少なくとも若干へ通知する装置
をも具備する請求項1に記載のコンピュータプロセッサ。

4．群内の全ての演算の引退が可能になった時にのみその群の何れかの演算の引退が進行されるように発機演算をグループ化する装置
をも具備する請求項3に記載のコンピュータプロセッサ。

(2)

5．少なくともそれらの演算が未済ではあるが完遂はしていない期間の間メモリ書き込みを保留する装置と、

それらの演算が遂中した時に、保留されている書き込みを流出させる装置と、

それらの演算が引退した時に、緩衝されている書き込みのキャッシュまたはメモリへの配置を完了させる装置

をも具備する請求項3に記載のコンピュータプロセッサ。

6．書き込みデータが流出されるか、またはキャッシュもしくはメモリ内に配置される前に、保留されている書き込みデータを後続の読み出し演算へ戻す装置をも具備する請求項5に記載のコンピュータプロセッサ。

7．上記実行装置がm個のプログラマ可視のレジスタを含む命令集合アーキテクチャを有する入力命令に応答し、少なくとも若干の演算がこれらのレジスタの1つを変更し、

未済のレジスタ変更演算の数をnに制限する装置と、

少なくとも（m＋n）個の物理レジスタと、

プログラマ可視のレジスタを物理レジスタへ写像する装置

をも具備する請求項1に記載のコンピュータプロセッサ。

8．物理レジスタを変更した演算が成功裏に引退してしまうまで物理レジスタが再使用されないことを保証する装置と、

仮想から物理への写像を異常状態が検出された前の実行状態に復元し、従ってプログラマ可視のレジスタの内容を復元する装置

をも具備する請求項7に記載のコンピュータプロセッサ。

9．特定の型の書き込みを処理するためにそれぞれ割り当てられる複数の書き込みバッファ手段と、

各書き込み待ち行列に最高優先順位を選択する装置と、

最高優先順位待ち行列を選択する装置と

を設けることによってインクロック回路及び性能を向上せしめたことを特徴とするコンピュータプロセッサ。

10．命令を含む入力流に応答してこの入力流内の命令を一連の演算に変換する装置と、

これらの演算の少なくとも若干をそれぞれが実行できる複数の機能ユニット、

これらの演算を機能ユニットの少なくとも若干に通信する（このように通信される演算を未済演算と言う）装置と、

タグを各未済演算に順次に割り当てる装置と、

機能ユニットによる演算の終了に関する情報を各未済演算毎に維持する装置と、

各機能ユニットに結合され、通信されて来た各演算についてそれが何時終了したかを決定し、この終了情報をその演算のタグと共に前記維持装置に通信する装置と、

最古の未済演算を決定する装置と、

最古の未済演算の表示を機能ユニットに通信する装置と、

少なくとも最古の未済演算に関する終了情報に応答し、その演算の終了情報が全ての機能ユニットがその演算を正常に終了させたことを示している場合にのみ、その演算の引退を許容する装置と、

少なくとも最古の未済演算の引退に応答し、このように引退した演算が最早未済ではないことを表すように最古の未済演算の表示を更新する装置と

を具備することを特徴とするコンピュータプロセッサ。

11．所与の未済演算が異常に終了したことの情報に応答して、流出させるべき演算の数を指定する打ち切りタグを機能ユニットへ通信する装置と、

各機能ユニットに結合され、打ち切りタグによって指定された全ての未済演算を流出させる装置と、

打ち切りタグによって指定された演算から未済演算の指定を削除する装置と、

流出した最古の演算のタグに等しい値から始まるさらなるタグの割り当てを前記タグ割り当て装置に開始させる装置

をも具備する請求項10に記載のコンピュータプロセッサ。

12．命令を含む入力流に応答してこの入力流内の命令を一連の演算に変換する装置と、

これらの演算の少なくとも若干をそれぞれが実行できる複数の機能ユニット、

これらの演算を機能ユニットの少なくとも若干に通信する（このように通信される演算を未済演算と言う）装置と、

未済演算の数を所定の最大数に制限する装置と、

タグを各未済演算に順次に割り当てる装置と、

機能ユニットによる演算の終了に関する情報を各未済演算毎に維持する装置と、

各機能ユニットに結合され、通信されて来た各演算についてそれが何時終了したかおよびその終了は正常であったかを決定し、この終了情報をその演算のタグと共に前記維持装置に通信する装置と、

未済演算の表示を機能ユニットに通信する装置と、

機能ユニットからの終了情報に応答して、正常に終了した演算を順番に引退させる装置と、

所与の未済演算が異常に終了したことの情報に応答して、少なくともこの所与の未済演算と、より遅い全ての未済演算とを流出させることを機能ユニットに命令する装置と、

機能ユニットに結合され、命令装置によって指定された全ての未済演算を流出させる装置と、

流出した最も早い未済演算のタグに結合されていたタグから始まるさらなるタグの割り当てを前記タグ割り当て装置に開始させる装置

を具備することを特徴とするコンピュータプロセッサ。

13．未済演算の少なくとも若干をそれぞれが実行できる複数の機能ユニットを含むコンピュータプロセッサにおけるパイプライン化演算を制御する方法であって、

割り当てられたタグを検査することによってなつの未済演算の相対年齢を決定できるように順序を付けたタグの共有の一員であるタグを各未済演算に割り当てる段階と、

所与の未済演算が完了する時点を決定する段階と、

未済タグの独学さを決定するように未済演算の数を制限する段階と

を具備することを特徴とする方法。

14．上記演算の少なくとも若干が分岐演算であり、

未済分岐演算の結果を予測する段階と、

未済分岐演算の誤った予測を検出する段階と、

誤って予測された分岐演算の結果として発行された全ての未済演算を流出させる段階

をも具備する請求項13に記載の方法。

15．上記割り段階がm個のプログラマ可視のレジスタを含む命令集合アーキテクチャを有する入力命令に応答し、少なくとも若干の演算がこれらのレジスタの1つを変更し、

未済のレジスタ変更演算の数をnに制限する段階と、

少なくとも（m＋n）個の物理レジスタを準備する段階と、

プログラマ可視のレジスタを物理レジスタへ写像する段階

をも具備する請求項13に記載の方法。

16．物理レジスタを変更した演算が成功裏に引退してしまうまで物理レジスタが再使用されないことを保証する段階と、

仮想から物理への写像を異常状態が検出された前の実行状態に復元し、従ってプログラマ可視のレジスタの内容を復元する段階

をも具備する請求項15に記載の方法。

JAPANESE LAID OPEN PATENT PUBLICATION

H4-503582 (1992)

(19) Japan Patent Office (JP)			(11) Kohyo No.	H4-503582
(12) Kohyo Tokkyo Koho (A)			(43) Kohyo Date	June 25, 1992
(51) Int. Cl.[5]	Identification Code		In-House Reference. No.	
G 06 F 9/38	370	A	8725-5B	
15/16	370	Z	9190-5L	

Preexamination request | No examination request | (totally 30 pages)

(54) Title of the Invention

DISTRIBUTED PIPELINE CONTROL SYSTEM OF COMPUTER AND METHOD THEREOF

(21) Application No.	PA H2-504389
(86) (22) Date of Filing	February 21, 1990 (Heisei 2)
(85) Submission Date of Translation	August 23, 1991 (Heisei 3)
(86) International Application	PCT/US90/00938
(87) International Publication No.	WO90/10267
(87) International Publication Date	September 7, 1990 (Heisei 2)
(32) Priority Date	February 24, 1989 (Heisei 1)
(33) Priority Claim Country	U.S.A. (US)
(31) Priority Claim No.	315,358

(72) Inventor **Harold L. McFarland**
4750 Applewood Drive
San Jose, California 95129, USA

(72) Inventor **Divitt R. Steires**

	830 Sunnywell Datosher Way Sunnywell, California 94087, USA
(72) Inventor	**Kolbin S. Van Dike** 45770 Kuga Court Flemont, California 94539, USA
(72) Inventor	**Shullenik Meter** 3164 Mount Isabelle Court San Jose, California 95148, USA
(72) Inventor	**John Gregory Faver** 5246 Raysan Court San Jose, California 95124, USA
(72) Inventor	**Dill R. Greenley** 1744 Letchud Drive San Jose, California 95124, USA
(72) Inventor	**Robert A. Kalgnoni** 1594 Moore Lane San Jose, California 95129, USA
(71) Applicant	**Nexgent Microsystems** 2202 North First Street San Jose, California 95131, USA
(74) Agent	**Hiroo SUZUKI**, Attorney

(81) Designated Countries	AT (wide area patent), BE (wide area patent), CH (wide area patent), DE (wide area patent), DK (wide area patent), ES (wide area patent), FR (wide area patent), GB (wide area patent), IT (wide area patent), JP, KR, LU (wide area patents), NL (wide area patent), SE (wide area patent)

[Claims]

Claim 1.

A computer processor, characterized by the fact that it provides:

a unit for issuing a series of operations achieving an unfinished operation, respectively if issued,
multiple functional units capable of executing at least some of unfinished operations, respectively,
a unit for allocating tags comprising a member of an ordered tag assembly to each unfinished operation so that the relative ages of two unfinished operations can be determined by inspecting the allocated tags,
a unit for determining the time of completing the given unfinished operations, and
a unit for limiting the number of unfinished operations to ensure the uniqueness of the unfinished tags.

Claim 2.

The computer processor referred to in Claim 1, in which the above limiting unit allows making at most n operations to be unfinishable at a time, wherein the above tags are issued in order over some range in a range more than or equal to 2n, and the relative ages of two unfinished operations can be determined by a comparison of the tags.

Claim 3.

The computer processor referred to in Claim 1, which has at most n unfinished operations at a time, indicating the tags as N-bit vector (wherein N is equal to or greater than n) with at most one bit set up, and
issues the tags so that an assembly of unfinished operations is indicated by adjacent groups of bits in the N-bit vector.

Claim 4.

The computer processor referred to in Claim 3, which also provides a unit for communicating the N-bit vector indicating that the assembly of unfinished operations be outflowed to the functional units in response to an abnormal state and a decision of outflowing the assembly of unfinished operations.

Claim 5.

The computer processor referred to in Claim 4, in which the above N-bit vector indicating the outflow of assembly is provided with a bit assembly which corresponds to the assembly of the outflow operations.

Claim 6.

The computer processor referred to in Claim 3, which also provides a unit for communicating the N-bit vector indicating a withdrawal of the assembly of unfinished operations to the functional units in response to a normal end of at least the oldest unfinished operation and a decision on withdrawal of the assembly of unfinished operations.

Claim 7.

The computer processor referred to in Claim 6, in which the above N-bit vector indicating the withdrawal of an assembly is provided with a bit assembly corresponding to the assembly of withdrawn operations.

Claim 8.

The computer processor referred to in Claim 1, in which the above functional units are realized in

multiple semiconductor chips.

Claim 9.

The computer processor referred to in Claim 1, which also provides a unit for outflowing all unfinished operations with tags issued later than a given tag in response to an abnormal state.

Claim 10.

The computer processor referred to in Claim 9, which also provides a unit for grouping adjacent operations so that the outflow of any operation in a group causes the outflow of all operations in that group.

Claim 11.

The computer processor referred to in Claim 1, with at least some branch operations in the above operations also provides:
a unit for predicting the results of unfinished branch operations,
a unit for detecting mis-predictions of the unfinished branch operations, and
a unit for outflowing all unfinished operations as the results of mis-predicted branch operations.

Claim 12.

The computer processor referred to in Claim 1, which also provides:
a unit for determining the oldest unfinished operation and
a unit for giving notice of a successful withdrawal of the operations to at least some of the above functional units by preparing a tag marking a boundary between the unfinished operation and the withdrawn operation.

Claim 13.

The computer processor referred to in Claim 12, which also provides:
a unit for grouping adjacent operations so that the withdrawal of any operation in a group is performed only when the withdrawal of all operations in the group has become possible.

Claim 14.

The computer processor referred to in Claim 12, which also provides:
a unit for buffered intermediate memory writes in such a period that at least their source operations are unfinished but not withdrawn,
a unit for outflowing the buffered writes when those source operations outflow, and
a unit for completing the arrangement of buffered writes in a cache or memory when those source operations are withdrawn.

Claim 15.

The computer processor referred to in Claim 14, which also provides:
a unit for returning the buffered write data to subsequent read operations before the write data are outflowed or arranged in a cache or memory.

Claim 16.

The computer processor referred to in Claim 1 wherein the above issuing unit responds to an input instruction with an instruction set architecture containing m programmer visible registers and at least some operations change one of these registers, and which also provides:
a unit for limiting the number of unfinished register change operations to n,
at least (m + n) physical registers, and

a unit for mapping the programmer visible registers into physical registers.
Claim 17.

The computer processor referred to in Claim 16, which also provides:
a unit for ensuring that the physical registers are not re-used until the operations changing the physical registers are successfully withdrawn and
a unit for restoring the mapping from virtual to physical to a precedent state when detecting an abnormal state and therefore restoring contents of the programmer visible registers.

Claim 18.

The computer processor referred to in Claim 17, in which the register change is allowed except for an order assigned by a programmer.

Claim 19.

The computer processor referred to in Claim 17, which also provides:
a unit for examining effective bits of all physical registers requested as an input to the operations, and
a unit for delaying the execution of operations of at least one physical register where the effective bits are cleared.

Claim 20.

The computer processor referred to in Claim 1, in which only the execution of interlocked operations is delayed by specific functional units directly affected when an interlock is requested.

Claim 21.

The computer processor referred to in Claim 1, in which some or all functional units provide

notification of the end of some or all operations, said end providing information sufficient for determining the tags of the completed operations, and providing information sufficient for determining an abnormal state of the highest priority order (if any) detected by the functional units in processing the operations.

Claim 22.

The computer processor referred to in Claim 21, which also provides:
a unit for attaching a priority order to end notifications provided by the functional units for each unfinished operation,
a unit for selecting the oldest abnormally completed operation, and
a unit for determining the correct response to the highest priority order of the oldest abnormally completed operation.

Claim 23.

The computer processor referred to in Claim 21, in which
notification of end from the functional units can be provided by an order different from the order of issuing the operations.

Claim 24.

A computer processor, characterized by the fact that it improves the interlock bypass and related properties by providing:

a unit for selecting multiple write buffer queues allocated to process specific types of write, respectively and the highest priority order for each write queue, and
a unit for selecting the highest priority order queue.

Claim 25.

The computer processor referred to in Claim 24, in which the above write is allowed in a out of order manner.

Claim 26.

A computer processor, characterized by the fact that it is provided with:
a unit for transforming instructions in an input stream containing the instructions to a series of operations in response to the input stream,
multiple functional units capable of executing at least some of these operations,
a unit for communicating these operations with at least some of the functional units (hence the communicated operations are referred to as unfinished operations),
a unit for allocating tags to each unfinished operation in order,
a unit for maintaining information relating to the end of operations by the functional units for each unfinished operation,
a unit for determining when each operation combined and communicating with each functional unit is completed and for providing notification of the end information to the maintaining unit together with the operation tag,
a unit for determining the oldest unfinished operation,
a unit for communicating the indication of the oldest unfinished operation with the functional units,
a unit for permitting the withdrawal of the oldest unfinished operation in response to the end information for at least the operation of only when the end information of the operation shows that the operation is normally completed by all the functional units, and
a unit for renewing the indication of the oldest unfinished operation in response to the withdrawal of at least the operation to indicate that the operation thus withdrawn is not the earliest unfinished one.

Claim 27.

The computer processor referred to in Claim 26, characterized by the fact that it is provided with:
a unit for communicating to the functional units, the truncation tag assigning a group of operations to be outflowed in response to information that the given unfinished operations have been abnormally completed,
a unit for outflowing all the unfinished operations combined with the functional units and assigned by the truncation tag,
a unit for deleting the assignment of the unfinished operations from operations assigned by the truncation tag, and
a unit for commencing the allocation of tags started from a value equal to the tag of the outflowed oldest operation in the tag allocation unit.

Claim 28.

The computer processor referred to in Claim 27, in which the truncation tag is equal to the tag of outflowed oldest operation.

Claim 29.

A computer processor, characterized by the fact that it is provided with:
a unit for transforming instructions in an input stream containing the instructions to a series of operations in response to the input stream,
multiple functional units capable of executing at least some of the operations,
a unit for communicating the operations to at least some of the functional units (hence the communicated operations are referred to as unfinished operations),
a unit for limiting the number of unfinished operations to a prescribed maximum,
a unit for allocating tags to each unfinished operation in order,

a unit for maintaining information on the end of operations made by the functional units for each unfinished operation,
a unit for determining when each operation combined and communicated with each functional unit is completed and communicating the end information to the maintaining unit together with the operation tag,
a unit for communicating the indication of unfinished operations with the functional units,
a unit for withdrawing normally completed operations in order in response to the end information from the functional units,
a unit for instructing the outflow of at least the given unfinished operation and all later unfinished operations to functional units in response to information that the given unfinished operation is abnormally completed,
a unit for outflowing all the unfinished operations combined with the functional units and assigned by the instruction unit, and
a unit for starting the allocation of tags started from tags combined with the outflowed earliest unfinished operation in the tag allocation unit.

Claim 30.

The computer processor referred to in Claim 29, in which the indication of the unfinished tag is a tag for the oldest unfinished operation.

Claim 31.

A method for controlling pipelining operations in a computer processor containing multiple functional units capable of executing at least some of the unfinished operations, respectively is characterized by the fact that it is provided with a step for allocating tags being one member of an ordered tag assembly to the unfinished operations so that the relative ages of two unfinished operations can be determined by inspecting the allocated tags.

a step for determining a time of completing the given unfinished operations, and
a step for restricting the number of unfinished operations to ensure the uniqueness of the unfinished tags.

Claim 32.

The method referred to in Claim 31, in which the above restricting steps at most n operations at a time allowed to be unfinishable, the above tags are issued in order over a range where they are greater than or equal to 2n, and the relative ages of two unfinished operations can be determined by a comparison of the tags.

Claim 33.

The method referred to in Claim 31, which also is provided with steps for indicating any abnormal state under which the functional units are performed and a step for outflowing all unfinished operations with tags issued later than a given tag.

Claim 34.

The method referred to in Claim 33, which also is provided with a step for grouping adjacent operations so that the outflow of any operation in a group causes the outflow of all operations in that group.

Claim 35.

The method referred to in Claim 31 wherein at least some branch operations in the above operations have:
a step for predicting the results of unfinished branch operations,
a step for detecting mis-predictions of the unfinished branch operations, and
a step for outflowing all unfinished operations as the results of mis-predicted branch operations.

Claim 36.

The method referred to in Claim 31, which also is provided with a step for determining the oldest unfinished operation and a step for giving notice of the successful withdrawal of an operation to at least some of the above functional units by preparing a tag for marking a boundary between the unfinished operation and the withdrawn operation.

Claim 37.

The method referred to in Claim 36, which also is provided with a step for grouping adjacent operations so that the withdrawal of any operations in a group is performed only when the withdrawal of all operations in that group becomes possible.

Claim 38.

The method referred to in Claim 36, which also is provided with:
a step for buffering intermediate memory writes in a period in which least their source operations are unfinished but not withdrawn,
a step for outflowing the buffered writes when those source operations outflow,
a step for completing the arrangement of the buffered writes into a cache or memory when those source operations are withdrawn.

Claim 39.

The method referred to in Claim 38, which also is provided with:
a step for returning the buffered write data to subsequent read operations before the write data are outflowed or arranged in a cache or memory.

Claim 40.

The method referred to in Claim 31 wherein the above issuing step responds to an input instruction with an instruction set architecture containing m programmer visible registers and at least some operations change one of the registers, which also is provided with
a step for limiting the number of unfinished register change operations to n,
a step for preparing at least (m + n) physical registers, and
a step for mappping the programmer visible registers into the physical registers.

Claim 41.

The method referred to in Claim 40, which also is provided with:
a step for ensuring that the physical registers are not re-used until the operations changing the physical registers are successfully withdrawn and
a step for restoring the mapping from virtual to physical to a precedent state when detecting an abnormal state and therefore restoring the contents of the programmer visible registers.

Specification

Distributed Pipeline Control System and Method of Computer

Background of the Invention

The present invention generally relates to a computer and, more specifically, to the efficient pipeline control of a computer.

Deep pipelines are necessary for realization of single cycle for a complicated instruction set computer (CISC) architecture. For complicated privileges and protection inspection as well as a powerful memory control system supported by the CISC architecture, if common pipeline techniques are combined, it becomes extremely complicated. Pipelines must contain the effect of multiple chip boundary intersections in present art. High-level VLSI integration in which these intersections should be excluded as far as possible is selected. If the system only contains a smaller number of devices, there is no signal pin sufficient for allowing a dedicated bus for all purposes to run. This means that a bus must be used for multiple purposes and therefore a centralized control and scheduling machine design process becomes extremely complicated.

Outline of the invention

The present invention provides a pipeline control system dispersed in all functional units in a processor. Each unit limits its own interlock and pipeline timing. The timing is not accurately monitored in a centralized control system. Because the functional units are autonomous, it is unnecessary to accurately know the details of "how all the units except for itself can process each instruction?", thus the complicated simulation of pipeline timing is sharply reduced. The present invention supports distributed control of the pipelines by enabling a "back-out" of changes for mechanical states which must not be generated. In the present invention, use is made not of a very complicated special pipeline control logic, but of a generalized technique, thereby correct actions for

pipelines are more promising. Distributed control combining unnecessary changes with a back-out capacity can provide important advantages in terms of properties in the area of a parallel processing of out of order execution, penalty cycle, and instructions in functional units and among functional units. Additional costs and complexity for realizing these capacities is very small.

Specifically, pseudo-operations (p-op or p-ops) with tags corresponding to decoder logics are issued to multiple functional units capable of executing the p-ops independently. P-ops up to any time can be unfinished. The tags are issued in order so that the relative ages (times) of two unfinished p-ops can be determined. In a specific embodiment, the tags are issued over a range of at least 2n and recycled. In this range, it is sufficient for enabling to determine the relative ages by simple subtraction. In this embodiment, it is allowed that 16 tags be issued and 7 p-ops be unfinished. P-ops are withdrawn in their order of issuance. The p-ops can be withdrawn only when they are completed. Namely, when they are usually completed by all correlated functional units. In some cases, completed p-ops usually qualified for withdrawal are kept unfinished until one or more adjacent young p-ops are also completed. Because notification of the tags of the oldest unfinished operations is made to the functional units, the units can determine a time at which they can be changed so that a mechanical state cannot be cancelled.

Unfinished p-ops are truncated if they are abnormally completed by the functional units. Old p-ops can also be truncated if the p-ops whose withdrawal is abnormally completed are successfully completed. The tags of truncated oldest unfinished p-op are also communicated to the functional units, enabling the truncation of execution when the detour and machine of an expected program are returned to a detour point.

In case of an instruction set architecture which permits m units of programmer visible (virtual) registers to exist and up to n p-ops for register changes are unfinished, at least (m + n) physical registers are provided. A machine for mapping the virtual registers into the physical registers is provided. Because the mapping is changed, as the destination of each p-op for changing the virtual registers to employ physical registers which have not been used so far, the values of old virtual

registers can be kept in the physical registers where they have been mapped before. If the physical registers substituted in mapping is re-used in order, there are physical registers sufficient for ensuring that the p-op mapped into some virtual register is probably withdrawn or truncated until some physical register must be re-used. Because a set of pointers limiting the virtual vs physical mapping and a list of available registers are maintained for every n latest issued p-ops, the virtual registers can be returned to precedent values without truncating unfinished p-ops and moving data among the registers.

Another technique enabling to return the state of the processor requires the use of a write queue. In a period in which at least the transmission p-op (p-op producing addresses and data) is unfinished, a write reservation queue buffer is written into a memory or a data cache. A write reservation queue entry is output to the memory only when the processing passes a point at which the back-out of memory write may be required. If the transmission p-op is truncated, the queue entry is deleted from the queue. When a young read p-op searches an access to a memory position written by an unfinished write p-op, the data stored in the write reservation queue are read and fed to the p-op. If the write p-op is withdrawn, the read p-op acquires the correct data without its withdrawal. On the contrary, if the write p-op is truncated, the young read p-op is also truncated, and the mechanical state is successfully return to a point before the write.

A further understanding of the essence and advantages of the present invention are realized by reference to the following description and drawings.

Brief description of the drawings

Fig. 1 is a block diagram of computer system incorporated with the present invention,

Fig. 2 is a high-level block diagram of a decoder (DEC),

Fig. 3 is a detailed block diagram of DEC,

Fig. 4 is also a detailed block diagram of DEC,

Fig. 5 is also a detailed block diagram of DEC,

Figs. 6A – B are block diagrams showing tracking of a specific sequence,

Figs. 7A – B are also block diagrams showing tracking of a specific sequence,

Fig. 8 is a schematic diagram showing the re-allocation of registers,

Fig. 9 is a block diagram of a memory and cache controller (MCC),

Fig. 10 is a block diagram of an integer execution unit (IEU),

Fig. 11 is a continuation of the block diagram of integer execution unit (IEU) of Fig. 10.

Brief description of the tables

Table 1 is a p-op bus format,

Table 2 is a physical address bus (PAdeBus) format,

Table 3 is a data cache bus (DIOBus) format,

Table 4 is a data exchange bus (DXBus) format,

Table 5 is an IEU terminal bus format,

Table 6 is an AP terminal bus format,

Table 7 is a sequence of p-op issuance and terminals.

Embodiment

Outline of system

Fig. 1 is a block diagram of CPU **10** incorporated with the present invention. The CPU (also called F86) is so designed as to execute an instruction set (macro-instruction) having exchangeability for an instruction set of Intel 80386 referred to in an *Intel 80386 Programmer =LΞΛэZ]=Manual* published in 1986 by Intel Corp., Santa Clara, California, USA. Blocks in the diagrams generally correspond to presently embodified and separated integration chips or chip groups. The CPU communicates with a memory controller, I/O device and external devices, most likely other CPUs via a system bus **11**. Reference numbers. shown in the lower part of functional units are understood to mean not these external devices but elements inside the CPU **10**.

An instruction decoder (DEC) performs the instruction incorporation, instruction decoding and pipeline control. The DEC **12** alternately simultaneously arranges up to three pre-fetches of a stream of instructions optionally and selectively. The DEC **12** contains a completely associative branch prediction cache (BPC) **13**. The BPC is an integrated structure and contains branch history data, physical branch target address and branch target buffer for each cache memory. If a branch instruction is decoded, the BPC examines information on the branch. A branch independent of a predicted direction is executed in a single cycle and pipeline bubbles are not produced.

A macro-instruction is selected from three instruction buffers or one instruction buffer in the BPC in each cycle. This macro-instruction is decoded, assembled into internak 96-bit decoded instruction word (also known as pseudo-op or instructions or operations) and dispatched to various functional units. Decoding of instructions is generally advanced at a single cycle rate. A tag for uniquely identifying each presently unfinished p-op issued by the DEC **12** is provided to the p-op in the mechanism. The tag can easily determine the relative ages of arbitrary tags issued in increasing order. A bus transaction contains transmission p-op tag. A functional unit makes the p-op, address, operand and their tags into a set (pair).

The DEC **12** assumes the responsibility for tracking the state of unfinished p-op, pipeline control, and, if necessary, a special processing call.

An address preparation unit (AP) **15** calculates an execution address, performs segment re-arrangement and, if required, realizes page memory control system, The AP contains a transformation index buffer (TLB).

An integer execution unit (IEU) **17** performs single cycle execution of almost all integer instructions. The IEU contains an array of 8 x 32 multipliers and accumulators and micro codes of multiplication and division instructions. The pipeline control architecture enables the performance of both or either of the parallel execution or out of order execution of IEU integer instructions.

A numerical processor (NP) **20** can be arbitrarily or selectively contained in the CPU. The IEEE floating decimal point standard is realized with high performance. The NP is integrated in the pipelines and also does not impart any special overhead in connection with the transfer of instructions or operands. Integer (IEU) and floating decimal point (NP) instructions are simultaneously executed.

A memory and cache controller (MCC) **25** assumes responsibility for controlling the instruction and data caches and realizes a cache coherence protocol. The MCC controls an interface to the system bus and supports high-speed single and block mode transfer between the cache and the memory. As described later, the MCC also contains a write reservation table for integers, floating decimal points and system write and contains a read-after-write short-circuited path.

An instruction cache subsystem contains a tag RAM chip (ITAG) **27** and a cache RAM chip (ICACHE) **30**. Each entry in the ITAG **27** contains address tags, effective bits and attention bits for corresponding lines in **30**. The attention bits indicate that the DEC chips can also have data from the cached lines in BPC. The ITAG **27** also contains an assembly of instruction stream address registers **31** and contains incorporation addresses corresponding one by one to unfinished flows even if three registers exist.

A data cache subsystem contains a tag RAM chip (DTAG) **32** and a cache RAM chip (DCACHE) **35**. The DTAG **32** contains address tags and line state bits for lines in the DCACHE **35**. The considered line states are miss, share read, owned clean and owned duty and supports a write back multi-processor cache coherence protocol (changed write once). The tag RAM is a dual port type and enables both the CPU and bus snoop cache look-up in a single cycle. A data cache inter-face (DCI) chip **37** interfaces the DCACHE **35** to the system bus **11**.

The functional units are packaged in a specially ordered ceramic PGA containing a power and grounding plane and a combined decoupling condenser. About 25% of the pins are subject to power and grounding. In the case of 0.8 μ to 1.2 μ process, the I/O delay is equal to an on-chip limiting path. The I/O is incorporated into the pipelines between the chips. Therefore a cycle time is not added to the mechanism. A common static RAM is used for the ICACHE **30** and DCACHE **35**.

Communications among various functional units is performed via multiple internal buses. A 64-bit IFETCH_DATA bus **50** for instruction incorporation, a 104-bit p-op bus **52** for communicating the issued p-ops to AP, IEU, MCC and NP, a 5-bit tag state bus **53** for communicating unfinished p-ops information to AP, IEU, MCC and NP, a 32-bit physical register bus PAdrBus **55** for communicating physical registers, a 64-bit (32-bit in each direction) data cache bus (DIOBus) **57** for data cache transfer, a 32-bit data exchange bus (DXBus) **58** for interchip exchange, a 64-bit bus for cache/memory renewal, and multiple end buses (i. e., an AP end bus **60**, an IEU end bus **62**, an NP end bus **63** and MCC end bus **65**) from the functional units to the DEC **12**) are contained in the functional units. Some of these buses are full-width, and some are semi-width (time multiplexing). Generally, conversations among the functional units are restricted to transactions which are fully limited in internal processor buses.

Details of the multiple buses will be described later. The use method of a standard CMOS style time multiplexing I/O suggests that the transfer occurs at a boundary between phase 1 (ϕ1) and phase 2 (ϕ2) of a system clock. For the ϕ2 transfer, a transmission chip must prepare effective data for an I/O

driver before the end of $\phi 1$. The effective data are fed by an I/O transmitter of the trans-mission chip in the successive $\phi 2$. The $\phi 1$ transfer is exactly contrary timing.

Tables 1 – 6 show a bus format of the p-op bus **52**, PAdrBus **55**, DIOBus **57**, DXBus **58**, IEU end bus **62**, and AP end bus **60**.

Outline of pipeline control system

The pipeline control of processor is spread over and distributed in the functional units. Centralized scheduling or score boarding of the pipelines is not performed. The DEC **12** observes some general resource constraints in the architecture and timely delays the issuance of p-op violating the resource constraints. The functional units assume the responsibility for scheduling their own internal operations. An interlock inspection is performed at a local level. In a deeply pipelined mechanism, special inspection at various stages of the pipelines causes significant difficulty in control. Careful precautions must be taken in each stage when a change of state is delayed while an exception of precedent instruction still cannot be detected in other stages. Exclusive control logic is general, and pipeline simulation must be carefully performed.

The processor uses several simple, general and powerful techniques to deal with this complexity. The DEC **12** issues decoded instructions (p-ops), and a functional unit processes addresses and operands without sticking to the special detection results provided by other functional units. As described above, the tags are allocated by DEC **12** when they are issued for the p-ops, and the DEC uses these tags to track the p-op.

The DEC **12** assumes responsibility for determining a time the execution proceeds over special points. The DEC **12** restores the mechanical state to a point immediately before a p-op where the exception occurs (trouble exception) or a successive point (trap exception) by using the technique described below.

As described above, the functional units have the end buses return to the DEC **12**. Signals on the buses indicate a time at which the p-op is completed (by a tag) and whether any exception (if any) is detected by the units. The DEC tracks whether any p-ops are unfinished in the mechanism, tracks resource constraints and then determines a time at which special processing must be started by using this information.

The DEC **12** returns the mechanical state to a special point in response to an abnormal end, calls a special handler and commences to issue any of a different instruction stream or a sequence of macro-instructions. The processor can return the mechanism to a specific state as a part of the response of a DEC to an abnormal end by using one or more of five common mechanisms, comprising the issuance of a truncation cycle, re-allocation of registers, use of write reservations, and the use of history stacking and serialization of functional units.

The DEC **12** issues a truncation cycle when the instruction issued by the DEC must be swept away. In the truncation cycle, a tag for identifying a boundary between an instruction which should permit the end and an instruction which must be expelled from the mechanism is fed to all of the functional units.

The register re-allocation is used to restore the state of common register files and segment register files and outflow changes carried out for instructions which must be truncated. The functional units have more physically available registers than those assigned by an instruction set. The DEC **12** maintains a set of pointers for mapping programmer visible (or virtual) registers into physical registers. The DEC in assembling decoded instructions substitutes an appropriate physical register number for a register assigned field.

When virtual registers are changed, the DEC allocates new physical registers, then changes the pointer set and uses the allocated register numbers as destination registers. Even after the execution of instructions, old physical registers still contain the changed values of the virtual registers. In order to back out the register changes, the DEC must restore the pointer set to the values existent prior to

issuing the instruction. The released physical registers are arranged at the end of a fully completed the free list so that the physical registers do not appear at the head of the free list until their contents are not needed. As described below, the DEC maintains the history stack of pointer values.

Data write waits until it is known that the write reservation list is used in the MCC 25 and that the write is not truncated. The MCC receives addresses and operands on the internal data buses, matches them with tags and, if it is safe to do so, performs an irreversible write.

The history stack is used to store and restore miscellaneous mechanical state like register re-allocation pointers, flag registers, and program counters.

In the case of a rarely changed mechanical state, the cost of the history stack of values is neglected. In such cases, the functional units performing the changes (then only the units) stops the processing, and the tags of the oldest unfinished instructions in the mechanism (fed from the DEC) are examined in each cycle to determine a time at which all of the old instructions in the mechanism are successfully completed. If this time is reached, old values of the mechanical state need not be reserved, and the functional units perform irreversible changes of the mechanical state.

A distributed pipeline control scheme combined with a capacity of backing out state changes enables the optimization of many properties.

Each functional unit can receive all of the p-ops, but actually process p-ops requiring processing in the unit. This is contrast to common pipelines in which instructions flow through all of the stages regardless of whether or not the stages perform a useful operation.

Moreover, each unit immediately performs operations if all input operands become available. P-ops which have not completed preparations for executing the operations immediately are stored in the p-op queue of the unit. If completed, the results are transferred to the next stage for performing further

processing, and the next operation is examined. A stage stops the execution only if the stage has nothing available for execution.

This behavior enables out of order execution among the functional units. For example, in the case of a memory write having an address production interlock, the AP probably cannot compute the memory addresses. However, the IEU can feed data, immediately perform an instruction, and subsequently continue to the next. The AP interlock need not prepare pipeline bubbles in other pipeline stages. Afterward, the IEU can delay the multiplication or wait memory operands. The AP has a chance to catch up with the IEU at this time.

From a viewpoint of specific functional units, this is not a complicated concept. The functional units locally make decisions and do not entirely notice if the instructions have become completely out of order. The pipeline control mechanism ensures that changes made by instructions executed out of order can be outflowed. The functional units do not perform special inspections.

Out of order execution among the functional units are freely generated as the result of a distributed decision made in a processor. Even in one functional unit, the instructions can be safely executed in a out of order manner. The IEU provides an example of internal out of order execution. The IEU examines the head of the instruction queue and locates out whether its execution preparation has been completed. If a data interlock prevents it from being executed immediately, the IEU examines the next young instruction and locates out whether its execution preparation has been completed. This process is continued until an executable instruction is found. A data interlock penalty is paid only when there exists no an available instruction whose executive preparation has been completed.

Even if the IEU pays a data interlock penalty, it does not means that the processor loses a cycle as a whole. Even the IEU delays, the IEU can catch up at a time in which an unnecessary instruction is issued. Finally, one or plural penalty cycles can be superimposed with one or plural penalty cycles from the AP 15.

In a special case of functional units which select for instructions to be executed out of order, this represents a parallel execution of the instructions in the functional units. Namely, this concept is applied to instructions requiring multiple cycles. A parallel execution of other single cycle instructions makes it possible for the multiple cycle instruction to have an effective throughput of one cycle.

A DCache miss is usually a total cache miss penalty, therefore the pipeline is stopped. The cache miss penalty is reduced until a range where the functional units can discover an operation capable of execution without using cache data. This is also true for a miss in the TLB of an AP chip. These cases usually have a fairly high number of penalty cycles and are different from other cases where it is difficult to completely superimpose them with useful operations.

Pseudo op bus format

The format of a p-op bus **52** is shown in Table 1. This bus is 52-bit in width and is a time multiplexed bus. The DEC **12** drives this bus alone and issues p-ops to AP, IEU and NP. The bus uses a standard CMOS style time multiplexed I/O.

Typically, one 386/387 macro-instruction is transformed to one p-op issued to correlated functional units by a DEC. In some cases, one macro-instruction brings about a sequence of issued p-ops. This p-op issued sequence is atomic, that is, the issuance of p-ops of one macro-instruction is not alternately arranged with the issuance of p-ops of another macro-instruction (or a special processing sequence).

In the case of typical macro-instructions, one p-op contains information sufficient for making it possible to perform necessary operations of the macro-instructions in all correlated functional units. They include memory operand address operation and segment, transmission and destination operand registers, ALU operation, operand size, operand path assignment, state flag change and p-op tags as well as assignment of both or either of correlated displacement and immediate data values. NP p-ops also assign micro addresses.

Almost all p-ops are transferred onto p-op buses by the use of the two clock phases (ϕ1 and ϕ2) in one clock cycle. ϕ1 is used for transferring almost all control information contained in the p-ops, and ϕ2 is used for transferring both or either of displacements or immediate values (with only a little miscellaneous bits of control information). In some cases of p-ops containing both displacement or prompt values (cannot backed to 52 bits), a second clock cycle is used for transferring the prompt values. The second clock cycle always immediately follows the first clock cycle. Displacements are transferred to ϕ2 of the first clock cycle, and the prompt values are transferred to ϕ2 of the second clock cycle. The DEC **12** drives the p-op buses in all the clock cycles. Usually, it is a normal p-op, but the DEC has not completed preparations for issuing a normal p-op or delivers an empty p-op in a cycle incapable of issuance instead.

The philosophy of encoding an information in a p-op is to feed the control information in such a form that it is not encoded at a time as early as possible in a clock cycle existing earliest or can be decoded quickly. This is especially true for the start of an operation in which the quickness in the functional units is critical and for the extraction of displacements and prompt values and the derivation of appropriate addresses and data operands. Although only control information which is not so critical is transferred in ϕ2, and generally, each functional unit can perform assembly/incorporation of operands from both registers and p-op and starts internal operations in the next ϕ1, etc.

As described above, almost all macro-instructions are transformed to a single p-op. This contains some more complicated microinstructions, which complexity must be processed in one of the functional units via micro-instructions (e. g., multiplications at POPA in IEU, AP). In a possible case, however, the complicated macro-instructions are transformed to a p-op sequence independently executed by a functional unit without noticing the overall sequence. In some cases, the p-op sequence is essentially necessary for the quantity or essence of the control information for which it is necessary to communicate the re-allocation of multiple registers (only one per p-op is allowed), multiple p-op tags requested by an AP for formation of an appropriate memory request or the renewal of multiple registers and flags by an AP.

These combinations can also occur in some cases of complicated macro-instructions. Namely, a p-op sequence is issued, and one of the functional units enters into micro-codes to execute the core part or all of the macro-instructions with the successive p-ops. For example, the first pop of a sequence is processed by an AP, an IEU, and an AP, goes into the micro-codes to perform further operations separately. These further operations correspond to the issued successive p-ops. Conceptually, the p-ops of sequence are executed independently by the functional units, which is also true as IEU characters in this case. However, the AP must globally know the p-op sequence for macro-instructions. Accordingly, in this case, the AP goes into the micro-codes and simply synchronizes with successive p-ops. The AP independently executes and ends the p-ops in appearance, but the AP uses only p-op tags and one or two fields of p-ops internally.

Two additional explanations of general properties are made for the issuance and recognition of p-ops by functional units. First, most p-ops do not wait to come into the p-op input queues provided by all of the functional units. As a result, each functional unit does not find and process all the p-ops or consumes a time. Generally, the p-ops are recognized by the AP and the IEU or the AP and the NP. Some p-ops should be found only by the AP, and one or two p-ops are recognized by all three of the functional units. Only the AP locates all of the p-ops.

Secondly, when some reasons exist for the DEC to enter special processing, the DEC issues corresponding p-ops even if there still exist unfinished precedent p-ops which perhaps require a truncation of newer special processing correlated to the p-ops. Generally, the DEC performs the minimum necessary self-control in the p-ops issuance to ensure appropriate operations from the viewpoint of macro-instructions.

In this connection, from a microscopic viewpoint (i. e., at the level of individual p-ops), there exist extremely few apparent limitations on a p-op sequence which can be issued by a DEC or on the timing of their issuances. Accordingly the functional units can be obtained by making a small assumption. This particularly applies to the fact that a small assumption on truncation of the p-ops can

be made. Only the maximum total number of allowed unfinished p-ops and the maximum number of allowed unfinished NP p-ops as well as the most fundamental constraints, such as a guarantee about whether the p-ops are active/unfinished at any time, are clear.

There exists one aspect valuable for a simple explanation of ensuring an appropriate macroscopic instruction execution. Some p-ops change the programmer visible state, which does not support the back-out capacity after the F86 microarchitecture is changed by the p-ops. Conceptually, some rest of the functional units is needed so that the DEC can ensure the permanent execution of the p-op before all of the p-ops are executed. This is not carried out by such a common technique in which the DEC delays the issuance of the p-op (and all of the successive p-ops) until all of the functional units reach the state of rest. As a substitute, this is carried out by a (functional unit) reference localized only by units which require a rest of a given p-op. The DEC can issue the p-op and successive p-ops while performing the necessary extent of rest by correlated functional units. Moreover, units not included in the rest can completely continue to execute successive p-ops.

Outline of DEC, pseudo op tracking and issuance control

If each pseudo op (p-op) is issued from a DEC onto P-Op buses, the appropriate functional units (AP, IEU, NP) can wait for it. Next, each functional unit processes a stream of p-op loosely combined with other units and, if the p-op is completed, notifies the end to DEC **12**. The DEC **12** shown by a block diagram in Fig. 2 comprises a front end **100**, a decoder **102** and a back end **105**. Fig. 3 shows the DEC front end, Fig. 4 shows the DEC decoder and Fig. 5 shows the DEC back end.

The DEC front end **100** assumes responsibility for incorporating instruction bytes and feeding them to the decoder. Instructions are fed from a BPC **13** or from one of three instruction buffers fed by an IFETCH_DATA bus **50**. The instruction bytes are fed to a rotation/shift logic **110** where the instructions are aligned based on information from a PC (program counter) register **112** (24 bytes at a time). Eight bytes are fed to the decoder **102**, which determines the instruction length, which is communicated to the PC logic **112**. If an instruction is longer than 8 bytes, 8 bytes are communicated in one cycle, and instructions over 8 bytes are communicated in the next cycle.

The front end logic **115** controls a stream stack **117** and feeds a stream address to an ITAG **27**. Up to two unfinished branches and accordingly three unfinished streams may exist. The control logic **115** issues an instruction request assigning the stream to be incorporated to an instruction stream address register **31** in the ITAG **27** and receives the effective bits identifying the stream. If the ITAG feeds the address, it increments an appropriate address register. The control logic **115** also receives a signal from a PADR monitor logic **120** for detecting the write into the stream of instructions relating to self-modification codes.

The DEC decoder **102** decodes the macro-instructions and issues all p-op sequences onto p-op bus **52**. The decoder receives the instruction bytes (macro-instructions) loaded in an instruction register **130** from the front end **100**. The macro-instructions are decoded by a decoding logic **132**, a p-op type decoding logic **135** transmits information on the p-op type to the front end and back end while an instruction length decoding logic **137** communicates with the PC logic **112** at the front end.

SA decoder p-op assembly logic **140** receives the p-ops from a decoder logic **132** and changes them in accordance with register allocation information from the back end. The p-ops are driven onto the p-op bus **52** and loaded in a p-op output queue **142**. Issuance is delayed by an issuance holding logic **145** based on a control signal from the back end.

The decoder **102** contains a sequencer **147** which controls the issuance when multiple p-ops generate a single macro-instruction. A decode holding logic **150** prevents processing when effective instruction bytes do not arrive from the front end. The decoder **102** allocates tags with the issuance of a p-op. The tags are issued and accordingly re-used by a circulation sequence, but only one p-op corresponds to the tags at a given time. The range of tags must be sufficiently large in comparison to the number of allowed unfinished p-ops so that the relative ages can be determined. If it is a range at least two times as large as the maximum number of unfinished p-ops, a determination is made possible by a simple subtraction.

The back end **105** continues to track all of the unfinished p-ops that float around the CPU. The issuance of p-ops must be properly controlled to ensure reliable actions (correlating to a tagging scheme of a CPU for controlling a p-op, address and data processing), mediate an abnormal state known by the end of functional units and then start appropriate actions. If the decoder issues the p-op, it is transferred to the back end with the information on the p-op. This is used for identifying right actions necessary for performing the aforesaid tasks.

The back end comprises tracking logic **160** which continues to track all unfinished p-ops and a holding state logic **165** which responds to the unfinished p-ops and controls the issuance of the subsequent p-ops by the decoder to constantly satisfy various constraints (described later) required by the correct and reliable actions of the CPU. The tracking logic **160** feeds information containing tags of the oldest unfinished p-op (OOTag) to the tag state bus **53**. The back end also comprises a truncation logic **170** which processes the truncation of p-ops, a register re-allocation logic **175** maintaining a pointer assembly array **177** and a the free list array **178** described later, and a tag production logic **179** controlling the tag state bus **53**.

A back end bus logic **180** receives end information from each functional unit and makes it possible for the tracking logic **160** and the truncation logic **170** to maintain the state of each unfinished p-op. Some p-ops are accumulated until some future time. Tracking of normal actions mainly exerts an influence on the issuance of subsequent p-ops. However, if notification of an abnormality is made from the functional units due to corresponding ends, the back end solves multiple abnormal ends of a given p-op and starts an appropriate response, which includes delivery of a truncation cycle to all other functional units (also including MCC) in order to return the state of the CPU to some p-op processing precedent state..

The tracking logic **160** and the truncation logic **170** contain registers storing specific information on all unfinished p-ops. These registers are organized as 8 same register assemblies attached with numbers 0 – 7 corresponding to 3 least significant bits of tags of the unfinished p-ops. Since at most 7 p-ops can be unfinished and are issued according to the tags, the relative ages can be determined

based on position numbers. The tracking logic **160** contains 8 state registers **190**, 8 end registers **192** and 8 p-op information registers **193** having correlation logic. The truncation logic **170** contains 8 response selection registers **195** having correlation logic, 8 priority order registers **197** and 8 end memory information registers **198**.

Each state register **190** stores a single state bit set up if a p-op having a tag corresponding to its position is unfinished. Each end register **192** stores one end bit per functional unit. This bit is set when the functional unit ends the p-op or the functional unit needs not to cause an action for the p-op.

Each p-op information register **193** stores 8 bits relating to the corresponding p-op, which contain the most significant bits of p-op tags operated by the functional unit, p-op types (e. g., floating decimal points, branch), branch prediction information and truncation group bits. Namely, "0" indicates that the p-op cannot be withdrawn in accordance with no final number, on the other hand, "1" indicates that the p-op cannot be truncated without truncating adjacent old p-ops having "0" in the truncation group bits.

The collection of state bits makes it possible to identify the oldest unfinished p-op. The position of p-op feeds the 3 least significant bits of the tag, and the information registers feeds the most significant bits. The state bits and the bits in the p-op information registers **193** make it possible for the holding state operations logic **165** to determine holding states, as described later.

The response selection registers **195** feeds information for which a response is necessary to the front end. The priority order registers **197** assign appropriate actions that should be taken for a response to multiple abnormal ends of a given p-op. The memory information registers **198** maintain detailed end information containing details of abnormal ends from the functional units acting on correlated p-ops.

In almost all cases except the case of generating the truncation, the functional units are independent of the state of the unfinished p-ops. A major exception is the MCC **25**, and MCC must know if it is safe to actually perform both or either of memory and an I/O write in a cache and an output to the

remainder of the system. In special cases, the AP and IEU must also know if it is safe to execute some p-ops. All of these requirements are satisfied by the back end which continuously issues the information onto the tag state bus **53** indicating OO tags and signal truncation for each clock cycle.

Tag state bus

The tag state bus **53** is a 5-bit bus, the signals of which are limited to only $\phi 1$. Although this is the case in almost all cycles, when the bit <5> is 0, the bits <4 .. 0> indicate the OO tag as a tag of the oldest unfinished p-op. When the bit <5> is 1, it indicates a truncation, and the bits <4 .. 0> indicates that the tag of p-op is returned to the origin and truncated. This is called a truncation tag (ATag). The back end **105** invalidates the issuance of next p-op of the decoder and issues one of two types of empty p-ops. When the tag state bus indicates that a p-op with a tag = i is the oldest unfinished p-op, it means that all old p-ops (i. e., p-ops with a tag < i based on 4-bit two complementary operations) are not the earliest unfinished ones regarded as being withdrawn. All young issued p-ops containing p-op (i) (i. e., p-ops with a tag ≥ i) are unfinished. Of course, this excludes p-ops which have been issued and subsequently truncated.

A p-op regarded to be unfinished means that it is still possibly truncated, and that it is actually defined by an operation using the back end **105** when determining the withdrawal of the p-ops. If the processing of the p-ops is completed by all the functional units, it is general to withdraw the p-ops as immediate as possible (based on their ends). However, there are various constraints when the p-ops can be actually withdrawn. Some of the details will be described below.

When the oldest p-op is withdrawn, the tag state bus indicates the p-op by advancing it from an indication OO tag = i to an indication OO tag = i + 1. The oldest unfinished p-op can be advanced in each and all clock cycles. Several p-ops can also be effectively withdrawn in one clock cycle by

jumping from OO tag = i to OO tag = i + n (however, $1 \leq n \leq 7$). If the unfinished p-ops do not exist,

the next tag issued by the tag state bus is indicated as the oldest unfinished.

The truncation till a p-op with a tag = i (p-op (i)) outflows all p-ops with a tag ≥ i (based on two complementary operations attached with a 4-bit sign), and the state of the CPU should be rolled back to a state between p-op (i-1) and p-op (i). This includes the issuance of the next p-op tag to be issued. In other words, ttruncation should outflow the p-op (i) and all young p-ops and restore the CPU to a state in which these p-ops are not apparently issued.

The truncation till tag = i can be generated at any time and need not be delayed until the p-op (i) becomes the oldest unfinished p-op. Such a truncation can also be generated when p-ops with a tag ≥ i exist locally. However, the truncation tags and tags of all unfinished p-ops ensure that all tag comparisons on the relative age still can be relied on (as a marginal note, e. g., if 7 unfinished p-ops exist and this truncation occurs, the truncation must be 1 greater than the tag of 7th (i. e., the youngest) p-op).

These outflow and roll-back of state must be performed by each functional unit in cycles to which the truncation is notified (schematically). This is necessary because the issuance of new p-ops perhaps is started in the next cycle. This is particularly true for a macro-instruction ⌈transfer of control⌋ in which the direction or type (for transfer of control to a distance) is mis-predicted.

In short, each functional unit must empty itself in one cycle and return the state of processing to normality until the end of this cycle.

Generally, in cycles following the truncation cycle, another truncation cycle can be generated, or a p-op can be issued (using many subsequent cycles), or a simple empty p-op can be issued (because the decoder has still not completed preparations for issuing the next p-op). If the next cycle following some truncation cycle is not another truncation cycle, the p-op tag indicating the oldest unfinished may be the same as the cycle preceding the truncation cycle or has a number of tags advanced until the return of truncated tag. In this final case, all precedent (old) p-ops withdraw after the truncation

and then, of course, all young unfinished p-ops are generated at a time in which they do not further exist .

Tag issuance

The following explanation relates to p-op tags, an outline of what they are and how the DEC **12** issues them. All tags are issued from the DEC as a part of all issued p-ops. Each tag is used by functional units for attaching the tags to addresses and data correlating to each p-op. If up to seven unfinished p-ops are allowed, at least 3-bit tags are necessary. This is further expanded to 4 bits by using one superior bit to simplify the comparison of p-op tags for relative ages. As explained below, if the tags are allocated, a comparison with 4-bit two complementary signs indicates the relative ages of two tags. Attention should be paid to the fact that only the least significant bits are necessary to clearly identify the p-ops.

Regarding the order of macro-instructions, all p-ops obtained from these instructions are issued in order and the tags are also allocated in order. All the 16 tag values are considered to be effective, and the tag order are defined as [next tag]: = (⌈ existent tag⌋ + 1) modulo 16. Accordingly, the above comparison on relative age is operated.

In instruction processing free of truncation, the above explanation is applied as it is. If the return truncation generates until a tag = i and the CPU state rolls back until immediately before the p-ops, the tag allocation is also returned to tag = i and reset. In order to continuously ensure the reliability of the relative age comparison, the DEC must issue new p-ops starting at tag = i from this time. Tags of the truncated p-ops are again effectively issued to the new p-ops. For example, this means that the truncation returned to a point earlier than the precedent truncation presents an effect which asks "does only the second truncation occur?"

More generally, a set of scenarios which are not presumably constrained in connection with

truncation cycles and p-op tag issuance can occur. For example, if p-ops (3 – 7) are unfinished, they are truncated until the p-op (5) and a tag 5-8 is issued, truncated until the p-op (6), truncated until a p-op (4), tags 4 - 5 are issued, truncated until the p-op (3), and more p-ops are issued, etc. This scenario perhaps is possible or not possible by depending upon the operation of the CPU and the functional behavior of the DEC, but the main point is the relation between the issuance of tags and truncations. As explained in the previous paragraph, the functional units should quickly become empty in the truncations, and return to a normal operational state, forgetting the truncations.

Withdrawal of p-ops

If the p-ops are processed by the functional units, notice of end is sent to the DEC via the end bus of the units, and the end of p-ops by the functional units is indicated. These are monitored and tracked by the back end to control a time of withdrawal of p-ops. There are special internal reasons why the back end delays the withdrawal of some p-ops, but there generally exist two issuances dominating the time of the withdrawal of the p-ops. In other words, they guarantee the appropriate CPU behavior in a normal environment and guarantee the appropriate truncation possibility of macro-instructions (and special sequences).

The most fundamental p-op cannot be withdrawn until all functional units provide notification of the (generally normal) end of the p-op. If the decoder of DEC issues some p-op, information on the type of p-op is also transferred to the back end. This includes functional units in which the p-op is probably processed and therefore the end is expected, after the end of the p-op. Based on this information, the back end withdraws the p-op as quickly as possible with other constraints as states.

In case of a single and then a short p-op sequence macro-instruction, if a trouble exception is detected in the p-op, the DEC must treat the truncations of all instructions (i. e., all their p-ops). This requires that the back end not withdraw any p-ops until all of them have been completed (at the normal end). If the p-ops are successfully completed, all of them are simultaneously withdrawn.

In the case of p-ops approaching to the maximum limit of 7 unfinished p-ops, notification should be provided that this approach to instruction truncations is undesirable. For example, if an instruction is a 7 p-op sequence, after the 7th p-op is issued, the DEC is effectively at rest while all of the 7 p-ops are completed before more p-ops are issued. In the case of a p-op sequence longer than 7 p-ops, an approach different in supporting an appropriate instruction truncations is absolutely necessary.

In some cases, processing can be accomplished via combinations which enable actually generating some memory writes by instructions. In some cases, it is also possible or acceptable to carry out inspections for detecting special troubles, which are not inspected until one of the p-ops, after the sequence if the inspection is not carried out, by using one or more special p-ops at the beginning of the p-op sequence. The present invention adds the inspections carried out by the first p-op (etc.) of the actual p-op sequence among the special up front inspections, but only one of the early p-ops can bring about the instruction truncations and ensure the trouble free execution of all later p-ops.

If approaches supporting the instruction truncations are used, only early p-ops should be used as they are unfinished until all p-ops are successfully completed. Specifically, it is indicated that only the first of many p-ops of the sequences should be processed by the DEC (i. e., back end) in these sequences, and remaining p-ops are not so constrained. Information in this regard is transferred from the decoder to the back end within the DEC each time each p-op is issued. In the case of many combinations of special up front p-ops and the first p-op of actual sequences sufficient for catching all special troubles, if these early p-ops are respectively completed, they can be immediately withdrawn. However, they may be accepted if the special p-ops do not exert a significant influence on the back-out of instructions (i.e., they do not change the programmer visible state).

In a final general examination on withdrawal of the p-ops, even though all p-ops of some sequence of some macro-instruction have been completed, if an early p-op has still not been completed, the completed late p-ops cannot be withdrawn. This essentially is another measure for seeing that the p-ops must be with-drawn in order. However, if an old p-op is completed and can be withdrawn, both the p-op and these late p-ops are simultaneously withdrawn. Fig. 7 shows the sequence of tag

issuance and end. Four points A, B, C, D are shown in the sequence, and the boundaries of 4 spaces are restricted. Fig. 4(A), (B) and Fig. 7(A), (B) show information stored in the registers of the tracking logic **160** and the truncation logic **170** at sequence points A – D, respectively. A single p-op or a group of p-ops are assigned as belonging to a truncation group. The truncation group comprises one or more p-ops which must have been totally completed to complete any p-ops. In other words, if one of p-ops in the truncation group needs to be truncated, all p-ops in the truncation group must be truncated.

P-ops (3, 4, 5) are issued in the first spacing, and p-ops (4, 5) belong to a truncation group (AG). Fig. 6(A) shows the information in tracks in the truncation logic register. Specifically, a p-op is issued, the p-op information is stored in a position corresponding to a tag number, a state register for the p-ops (3, 4, 5) is set up, and the p-ops are assigned as having been issued. It is indicated that truncation bits for the p-ops (3, 5) are set up, and the p-ops (4, 5) belonging to a truncation group p-op (3) represents the only number of some truncation group.

A p-op (6) is issued in the second spacing and notification is given of the normal end of p-op (3). As is evident in from Fig. 6(B), state bits for the p-op (6) are set in a state register **190**(6), AP end bits for p-op (3) are set in an end register **192**(3) and then written in a end memory register **198**(3).

P-ops (7, 8, 9) are issued in the third spacing, and p-ops (7, 8) belong to some truncation group. In this spacing, IEU indicates that the p-op (3) has been normally completed, AP indicates that the p-op (4) has been normally completed and then indicates that the p-op (6) has been normally completed. Fig. 7(A) shows that state bits for the p-ops (7, 8, 9) are set in state registers **190**(7), **190**(0), **190**(1), IEU end bits are set in end register **192**(3) and **190**(6), and AP end bits are set in end register **192**(4). The corresponding normal ends are written in end memory registers **198**(3), **198**(6) and **198**(4). Attention

should be given to the fact that the p-op (3) may be withdrawn, and the state bits in the state register **190**(6) are cancelled.

In the fourth spacing, there exist 7 unfinished p-ops being the maximum allowed as unfinished, therefore additional p-ops are not issued. In this spacing, AP indicates that the p-ops (5, 6, 7) are normally completed, and IEU indicates that the p-ops (4, 5, 6) are normally completed. However, the next AP indicates that the p-op (7) has been abnormally (e. g., page trouble) completed, and subsequently the IEU indicates that the p-op (7) has been normally completed. As a result, p-ops (4, 5, 6) may be withdrawn, and they are not indicated as the oldest unfinished p-ops. However, p-op (8) has been abnormally completed, therefore the p-op (7), as one member of a truncation group of p-op (8) and a p-op (9) issued after the p-op (8), must also be truncated. Accordingly, the truncation logic **170** issues an ATag on the tag state bus of 7 and gives notice to the functional units that the functional units (AP and IEU in this case) must be returned so that the p-ops (7, 8, 9) are not issued.

P-Op issuance constraints

The back end tracks unfinished p-ops and the p-op end of each functional unit, and the holding state logic **165** in the back end also controls the issuance of additional p-ops by using the state of unfinished p-ops. In order to ensure right comprehensive actions of CPU and actions of specific blocks of logic in specific functional units (specifically DEC, AP and NP), the back end continuously imposes various constraints on the maximum of unfinished p-ops of various types. If it reaches a limit imposed by these constraints in terms of actions, the back end sends holding state signals the decoder to control whether a p-op issued in the next cycle is delayed or not.

The back end produces nearly a half dozen of holding state signals and send them to the decoder to potentially delay the next p-op. The decoder use these signals and generates actual p-op decoder/issuance holding based on whether existent decoded/assembled p-op and notified holding states are applied or not. Each holding state corresponds to one or more (analogous) constraints. If there are any constraints, if the back end determines that the unfinished p-ops are the maximum, and one of these p-ops is about to be completed, it generates corresponding holding states.

In case of many constraints, it is ensured that the correlation type oldest unfinished p-ops are

completed first p-op. In the case of some constraints, the holding states are not simply based on all unfinished (i. e., not withdrawn) p-ops but based on unfinished and not completed p-ops. If some p-op completely ends, the p-op is independent of some constraints accompanied with earliest specific hardware of the functional units even if it remains unfinished over several cycles forward.

The back end is one of major generators presenting holding states to the decoder, but several sources of holding states exist elsewhere. It provides notification of a limitation of whether such holding states are applied or not applied to the issuance of an existing p-op. To completely generalize this for the p-op issuance control, the pseudo-op bus is said to be driven by either an effective p-op or an empty p-op (mostly with the truncation action) in each clock cycle. From the viewpoint of decoder, a decoder always issues an effective p-op so long as any of the followings does not occur.

1) truncation priority from the back end,
2) holding from the back end,
3) holding from BPC,
4) holding from VIB (virtual instruction buffer),
5) decoding with prefix alone, and
6) sending of second half of two cycles

Of course, 5) and 6) are produced by the decoder, and 4) and 5) are applicable only to the first p-ops of a macro-instruction sequence.

"Holding from BPC" occurs when the decoder locates an instruction [transfer of control] which decodes the next macro-instruction and which can be cached in the BPC. The decoder must attempt BPC access on the prediction information of some entry (target flow of a paired entry) for such an instruction. The BPC access to the transfer instruction of control generates and decodes the instructions. If the decoder cannot use the BPC access cycle, BPC holding is produced. If the access to BPC about the prediction information can be used and a miss occurs, the decoder can carry it out even if the BPC target flow access cannot be used. If bits generate and an access too both parts of BPC is unavailable, BPC holding is produced. If this is not the case, the decoder can carry it out by

using the prediction information, while the target flow of BPC entry is dumped into a new queue allocated by the transfer control instruction.

"Holding from VIB" occurs when the decoder decodes the next macro-instruction but does not receive all the necessary instruction bits (for the instruction length). The decoder which accomplished the transfer must detect that effective prefix bytes has at least effective operation code bytes or the VIB holding must be enforced. If mod r/m bytes are necessary even if based on pre-decoding of the operation code bytes, they are also presented or the VIB holding must be enforced anew. Moreover, if s-i-b bytes are necessary even if based on pre-decoding of mod r/m bytes, the same applies to the s-i-b bytes. If these bytes are to be effective, final instruction bytes (actual VIB words containing them) are examined (and all intermediate bytes are also suggestively examined), and if they are not effective (i. e., "bad" or "empty"), VIB holding is produced.

"Decoding with a prefix alone" occurs when the decoder decodes the next macro-instruction, but only prefix have been decoded to that point, so two prefixes are further decoded. A case of one prefix and a second empty byte is treated as "holding from VIB" until the second byte becomes not empty or the first prefix byte is consumed and treated as decoding with a prefix alone in which the VIB is advanced.

The "sending of second half of" occurs when the first cycle of p-op of two cycles is about to be issued. In this cycle, a special empty p-op is sent with an additional p-op information, and the deco-ding and formation of the next p-op are delayed.

"Holding from the back end" occurs if the decoder understands that it is not safe to immediately issue a p-op based on a signal of back end because of the type of issued p-op. All constraints of unfinished p-ops enforced by the back end are listed below.

1) totally 7 p-ops

2) 2 transfer control p-ops

3) one truncation group in a single stepping mode,

4) two p-ops accompanied with the re-allocation of segment registers, and

5) 0 further first p-ops after the rest of DEC.

The maximum of totally 7 unfinished p-ops are applied to all non-withdrawn p-ops. Generally, therefore the p-ops are not completed in order in the case of these constraints. However, the p-ops can be withdrawn in order only by the back end.

Although the maximum two unfinished transfer control p-ops are applied to all of the p-ops, this constraint is actually applied to a transfer macro-instruction of control and first p-ops of their p-op sequence. In the case of this constraint, the transfer control p-ops are emphasized when they are unfinished and not completed. Whether the p-ops have been completed but are still not withdrawn, is not important in connection with this constraint. When a mutual instruction incorporation page request is produced, even if two transfer control p-ops are not unfinished, the back end provides notification of the holding state depending upon how they are processed. However, when all unfinished instruction incorporation double-length words exist on an older sequential instruction stream, the impact on this constraint does not exist. Attention should be given to the fact that the IEU requires that the transfer control p-ops (p-ops containing IEU) be completed in order.

If p-op single stepping is made possible (according to the purpose a hardware device), p-ops of a truncation group are issued one at a time, and completed and withdrawn before the next group is issued.

Because of a re-allocation scheme used for segment registers, there exist only two unfinished p-ops containing the segment register re-allocation for data segment registers (i. e., DS, ES, FS, GS). This constraint does not apply to p-ops exclusive for segment register read-out and stored in both or either of the CS or SS. Its purpose is to ensure that the possibility of truncation is transferred to any and all of the segment registers storing the p-ops. Because the AP rest behavior has already been applied to

CS/SS memory p-ops, it is unnecessary to include memory into CS and SS.

If the DEC rest p-op is issued, the decoder can continue to further issue p-ops in order, but the deco-ding of the next macro-instruction must be delayed until some renewed control bit information is received from the AP by the back end. The control bits comprise various bits of Eflags which affect the macro-instruction decoding of the decoder and the p-op aeembly process. A p-op which causes a change in one or more Eflag bits on which the decoder depends must be treated as a DEC rest p-op. Thereby, a DEC copy of these bits is renewed before more decoding of macro-instruction occurs. The back end produces holding states to prohibit more macro-instruction decoding and the issuance of a first p-op.

All decoder holding states except for truncation invalidity are determined sufficiently earlier to enable determination of the next operating state of a decoder until a time at which the decoder must start the next decoding cycle (namely, controls are prepared, and access to a new active queue is obtained to generate new VIB contents and then is not delayed in performing pre-decoding to advance the existent active instruction queue). Since a p-op produced by the decoder is destroyed and substituted by an empty p-op, truncation invalidity is subsequently produced or may even not need to be produced. At the same time, the p-op is jammed into a new p-op sequence which should be produced by the decoder and vectorized by the back end. (note: one or more of jam and vector types exist for timing and vector address)

As described above, in the case of normal states notified by various units (inside DEC), the decoder does not dialog with actual holding signals from the functions, and the decoder does not receive these signals. Instead, units send the holding state signals which are combined (logically multiplied) with state signals communicating the type of p-op in the production. These signals are combined (logically added) with additional holding signals produced by the decoder to generate comprehensive decoder holding signals. This not only controls the p-op issuance and decoder state sequencing, but also sends them to other units and exerts an influence on the decoder state sequencing by the dialog with the decoder only.

Functional unit rest

When the functional units process o-ops, they must truncate changes relative to the programmer visible state and correlated state or ensure a back-out capacity. They include all commonly changed property critical states, general-purpose registers, fixed decimal point registers, and almost all segment registers, PC, and state flags. Other states, i.e., a special state without changes is rarely returned via a history stack or by using register re-allocation. Instead, these state are processed by limiting the time in which they can be changed with dominating (one or plural) functional units.

Essentially, in the case of arbitrary special registers, (one or plural) owners delay the performance of changes until the correlated p-op becomes the oldest unfinished p-ops. If so, the possibility of truncating p-op disappears because of another (earlier) p-op. Moreover, the reason this p-op which brings about this truncation can be considered as a cause should be examined. This is considered safe for performing changes (if the dominating/changing functional units detect a reason for the back-out of p-op, it must be do so even there is no any necessity so that the changes can be cancelled).

If some p-op is added to the AP and processed by other functional units, only the dominating functional units can give notice of abnormal end. All of the p-ops are limited and written so that other functional units can always provide notification of normal end. If two functional units dominate special registers together, they change their own copies, respectively, and the two units will probably always provide notification of a normal end for the p-op.

In any case, only functional units depending upon a special state changed by some p-op are given a rest. All other functional units processing the p-op behave normally. Essentially, the rest of some p-op occurs only on a localized basis and in a necessary place. As many CPUs as possible continue normal processing, and only p-op processing made by (one or plural) rest functional units are mostly decelerated.

Most special registers are dominated only by the AP, and almost all rest p-ops require a rest caused only by the AP. Many of them are p-ops of the AP alone, while the remainder are AP/IEU p-ops. A rest caused by an NP (all of the AP/NP p-ops) is due to changes to the three control registers by which it is processed. In case of a dual function unit rest, they are limited only to the existent AP and IEU. This occurs when some p-op change the direction flags of the Eflag register. Since both AP and IEU maintain the latest copies, AP and IEU are parallel and perform an independent rest.
Even if some functional unit rests in processing an arbitrary p-op, this does not necessarily mean that the unit rests immediately prior to processing the p-op. Particularly, if the AP rests, a part of the processing of the p-op can be performed prior to the rest. That necessary for the AP is only the rest at the point of changing a special register. If the rest is completed, the AP change to continue the processing.

The DEC can also perform a rest, but it is only slightly similar to the rest performed by other functional units. Following the issuance of DEC rest p-op, the DEC delays the assembly and issuance of some p-ops. This delay occurs until the DEC receives ⌈control bit renewal⌋ from AP. For a further explanation of the DEC rest, please see the preceeding paragraph.

In the case of a DEC rest, DEC copies of some special control bits are renewed in a manner similar to another case of receiving the control bits renewal from AP by DEC. This occurs with a change of one's own copy of the control bits made by AP. The copy held by DEC is not found as a master copy owned by DEC, but is found as a secondary copy maintained in the DEC by the AP. The DEC does not have the capacity of backing out the renewal for these bits. However, this not becomes a problem because the AP also has to change the master copy of these bits and does not send the control bit renewal before changing their own copy. The AP is required to rest, therefore the renewal of the control bit copy of the DEC is effectively delayed by AP until the correlated p-op becomes the oldest unfinished one.

Abnormal end processing

As described above, the back end monitors the end of p-ops of the functions and accumulates the state of all pending p-ops. The back end controls the withdrawal of p-ops (generally for abnormal end after a normal end is made by all correlated units) based on this information and exerts an influence when new p-ops are issued by the decoder. If the p-ops are completed and one or more abnormal ends are received, the back end assumes the responsibility for an appropriate response and then starts it at an appropriate time.

If the back end receives ends, including abnormal end, of given p-ops, it generally accumulates the ends until all of predicted ends are received. If an abnormal end exists, the back end does allow its p-op to withdraw. At this time, the back end starts an appropriate response. If plural abnormal ends exist, the back end takes preference and selects a response to the abnormal ends. Both sides of the abnormal ends are explained below.

Waiting prior to starting is carried out to minimize the complexity in design for processing in case dialogs produce abnormal end responses nested/replaced by earlier/old p-op abnormal end responses (detected and started afterward). If only a case of abnormal end bringing about a special start is so processed, a significant penalty is not imposed on the properties by twaiting.

A specific response started by the back end depends upon whether the abnormal end and old p-ops are determined or not. This is not clearly dependent on the p-op, and particularly is not clearly dependent on the operation codes of the p-op. The response often sends a truncation cycle with an appropriate flag (it unnecessarily needs to be a tag of an abnormal end p-op). In the truncation cycle or a cycle in which an empty is issued without coexistent truncation, the back end jams and vectorizes the decoder into a state in which the decoder continues decoding, along with operations for issuing a p-op. When special processing must be started, the decoder is vectorized in an appropriate p-op sequence assembling and issuing before it returns to the macro-instruction processing. The abnormally completed p-op may be contained in the truncation or withdrawn to be like normal depending upon the type of starting the exception.

In almost all cases in which the responding abnormal ends do not bring about special processing, if a p-op completely ends, a prompt response is started. In case of only small special abnormal ends, a response occurs immediately after the ends are received by the back end. These ends are not considered to be normal ends, but are helpful in case of more than that. These ends are not true ends because subsequent ends are predicted and the production of special abnormal ends are requested from functional units.

Responses to these cases are similar to aforesaid responses including the start of special processing and have a possibility of not only vectorizing to some appropriate p-op sequence and instead, returning to macro-instruction and vectorizing it. In other words, late p-ops in the p-op sequence are truncated, and the decoder continues decoding a macro-instruction flow from the next instruction (from an existent instruction queue or a different instruction queue). When few abnormal ends exist, it brings about both or either of a case where responses do not directly affect the decoder or a case where other operations are started inside the DEC.

IEU end bus

Table 5 shows the format of a 5-bit IEU end bus. This bus uses a standard CMOS style time-sharing I/O and provides notification of a normal complation of p-ops and two types of abnormal ends (special and mis-prediction branch direction). The bus feeds a 3-bit p-op tag and a 2-bit end Id to $\phi 2$. Because of the timing of the DEC decoder and p-op assembly pipelines, if the IEU end codes and correlated p-op tags are sent to $\phi 2$ - $\phi 1$ (i. e., 1 phase earlier) by time sharing, the DEC can immediately respond in a truncation cycle followed by a correct next p-op (from a correct next macro-instruction or from an appropriate special processing p-op sequence).

Generally, the IEU can end p-ops out of order (to numbers of issuance made by DEC) and probably ends them in such a way. There are some special cases, e.g., the relative serialization between two p-ops of same type must be maintained by IEU as long as the p-ops relate to the processing/execution

order. Generally, the execution order in these cases is decisive but the end order is not. For transfer p-ops of control in which IEU only locates (near) control transfer p-ops with states, to end in relative serial order is required by the IEU. If viewed from the DEC, it is absolutely unnecessary to process these p-ops in order.

With the IEU processing the p-ops, there are two cases of when they can be completed. 1) When a DXBus transfer is not required after the p-ops are executed, the p-op can be completed if the correct end is known. 2) When the p-op requires such a transfer after execution, the p-op can be completed if it is known that the transfer occurs or it is known that the transfer actually occurs. In both cases, the end can occur after these times. In other words, in case of 1), the p-op can be completed in the ALU action if the end is unstateally normal, or the p-op can be completed immediately after the ALU action are completed if the end depends upon the ALU action. In case of 2), the p-op can be completed if it is known that the IEU wins over a DXBus mediation relating to the transfer.

The following real end behaviors of IEU are predicted now based on common IEU pipelines, out-put queue timing and instruction, and timing of IEUTerm (i. e., $\phi 2$ - $\phi 1$). In the case of p-ops for which the results needs not to be transferred via the DXBus, the end is started in an ALU action cycle. In the case of almost all p-ops, it is an unstateally normal end, and a correct end of the transfer control p-ops is determined in the first part of the ALU cycle (also applied to an INTO instruction p-op). Sometimes, this end which cannot go out to end buses is waited for and notification is subsequently provided to the DEC (most immediately).

In the case of p-ops for which the results needs to be transferred to the DXBus, the end is started in a transfer cycle. In this case, if the end cannot go out immediately, it is also waited for and sent afterward.

In the case of p-ops relating to BOUND and REPed column macro-instructions capable of causing an abnormal end and which are dependent on the ALU action, which belongs the above 1), the above timing of case 1) does not operate. In these cases, the p-ops are treated so that the results must be

delivered on the DXBus anyway.

There are two reasons why the end is produced out of order. First, the IEU selects processing/execution p-ops out of order. Secondly, the IEU can end the p-ops further out of order in the execution order. If outlined, the IEU immediately ends in the case of p-ops, and 2) the IEU must first go forward onto the DXBus (because it mostly waits in a data output queue of the IEU in such a way). In the latter case, the p-ops end if they actually go forward onto the DXBus. In addition, if they exceed 1) and 2) ends temporarily, there is also a possibility of notifying some high superior order end (e. g., transfer end of control) before the waited end. (Of course, the relative serialization of transfer control p-ops must be ensured).

In any case, the IEU must complete processing before the IEU ends the p-ops independent of the transfer end of control. This includes p-ops which bring about the register renewal from the AP to the IEU or simply transfer memory operands to the registers. For the both types of p-ops, transmission operands must be received before they are completed. If they are explained in contrast to the behavior of the AP, in the case of various transfers and register renewals, the AP can be completed before receiving, which is effective register renewal (even if the results of combined registers perhaps is needed).

After the IEU responds to an abnormality detected in processing a p-op and provides notification of a abnormal end, it continues the processing of other p-ops. The IEU stops the processing of p-ops and waits for a response which should come to an abnormal end in some sense.

IEU end

An end shown in Table 5 is described below.

When a real end to be known does not exists, the end is not notified. End buses are effective in all clock cycles, and any bus must be always be indicated.

If an abnormality is not detected in processing some p-op, notification is give of a normal end. When a predicted branch direction is not right, the mis-prediction branch direction end must be notified regarding the transfer p-ops control (there must be near-control transfer with states). This is a substitute for a normal end in case of correctly predicted branch direction).

Abnormal ends are due to special reasons and are used for notifying exceptions defined by corresponding architectures, respectively. A division error is used on p-ops attaching a note with EUabort in p-op sequences of DIV and IDIV macro-instructions. A constrained inspection and an INTO overflow are used on EUabort p-ops of BOUND and INTO macro-instructions. An REP'ed instruction repetition stop end is notified regarding p-ops attaching a note with p-ops of a p-op sequence of REP'ed column macro-instructions, i. e., EUabort p-ops. If a test performed by the p-op indicates that the repetition of column macro-instructions should be stopped, notification is made of this instead of the normal end. Even if a p-op test indicates that the repetition of column macro-instructions should not be started (i.e., performance of repetition), this is applied. If an exception is detected under these states, notification is give of a normal end.

There is no possibility that the IEU detects multiple abnormalities in one p-op, therefore a relative priority order is not issued between IEU abnormal ends, but a priority order is issued to the end of other functional units. Since only one type of exception for a given p-op type in IEU can exist, the DEC truncation logic can uniquely identify a special type based on the p-op. The IEU abnormal ends are grouped in several groups based on the superior order recognized by DEC for AP and NP

abnormal ends. Almost all abnormal ends are grouped in intermediate superior order groups while the REP stop ends have low superior order.

A mis-prediction branch direction end is special in that it does not have a specific superior order fixed to all AP ends. Instead, an execution branch direction (predicted direction and rightness of prediction) combined with AP ends determines actions started by the DEC back end.

AP end bus

Table 6 shows the format of an AP end bus **60**. This bus uses a standard CMOS style time-sharing I/O and provides notification of a normal end and various abnormal ends of p-ops.
If an AP end code is sent by 1 phase earlier ($\phi 2$ - $\phi 1$) time sharing, the DEC can immediately respond in a truncation cycle followed by a correct next p-op (from the next macro-instruction, or from an appropriate special processing or another p-op sequence) because of the timing and p-op assembly pipelines of a DEC decoder. When the coding of end codes is important, the DEC can provide an ideal response time for the DEC, issues or truncates other o-ops or issues the correct next p-op. In other special cases, an effective special cycle exists in the response time. Namely, one cycle generates before the truncation cycle, and the correct next p-op follows the next cycle.

This special cycle for processing almost all abnormal ends is distributed among the back ends of the DEC to find which occurs or how it does, the decoder of DEC is jammed and vectorized and the decoding of the correct next p-op is started. In the case of a quick end, the back ends have restricted processing states. To support this quick processing can predict p-op tags to which the back end is correlated to the next end because the AP always end the p-ops in order.

The quick end is designed for the normal end of some p-op and such a state of control bit renewal (from AP to DEC) also indicating both or either of any selectively mis-predicted addresses and D bits. In case of the normal end, the back end having p-op tags and information on type of p-op under control must reflect this end in a holding state signal in the decoder and branch control logic. In case of the control bit renewal accompanied by both or either of any selectively mis-predicted address and D bits, the end buses enable to feed a renewal value for the control bits and subsequently the decoder continues the decoding of a macro-instruction instruction flow. If both or either of mis-predicted address and D bits are also indicated, the timing of this end effectively becomes same as all other non-quick ends.

As described above, the AP must end p-ops in order (to the order of p-ops issued by DEC). This is independent of the order that AP processes the p-ops, but there exists a constraint on the order that AP can process p-ops for other reasons. In all the cases, a p-op can be completed anytime after it is completed. However, somewhat similarly to the state of IEU, there exist two cases of the earliest time when the p-ops can be completed. Case 1 is a case where the p-ops do not require DXBus transfer after execution, if a correct end is known, the p-op can be completed. In the case 2 where the p-ops require such a transfer, if it is known that the transfer occurs, the p-op can be completed. In other words, if the reference (relative to abnormal end) and necessary inspection of all sys-tem memories are complete, the p-op can be completed. In the case 2, if it is known that the AP is successful in the mediation of DXBus or PAdrBus in relation to a transfer and the transfer occurs, the p-op can be completed. This includes a case where the reference transfer of the PAdrBus memory address is truncated because of TLB miss. This end cannot occur prior to whether the transfer is actually completed or not. Additional constraints/requirements provided by DEC and specified in some ends are described below.

Attention should be paid to the fact that except for the reception of general-purpose register renewals from an IEU, NP or memory, the AP can end a processed p-op before the renewals are received. The renewal essentially does not require more processing, and is simply stored into appropriate registers. Only the register interlock control may be renewed for communicating it. The AP ensure these renewals until a time when the correlated p-op completely ends and therefore before it withdraws. Of course, the AP still must properly track the register renewals predicted regarding the occurrence of truncation.

After the AP responds to an abnormality detected in processing some p-op and provides notification of an abnormal end, it properly ends the p-op. The AP can further interrupt processing the p-op depending upon the end. This behavior occurs after an abnormal end such that the DEC starts special processing to respond to it. In all other cases, the AP continues the processing.

AP end

The end shown in Table 6 is described below. For all abnormal ends indicating exceptions on p-ops, attention should be paid to the fact that bits <3 .. 0> of end Id directly correspond to an interrupt number of exception for which a processing should be started. Two exceptions for it are substitution ⌈decoder and common protection⌋ trouble code (i. e., 1111 010X) used in special cases. An ⌈interrupt abnormal end⌋ (code = 1111 1001) is also special in that a special processing does not occur and the DEC is interrupt instead.

When a real end to be notified does not exist, the end is not notified. When an abnormal end is not detected in processing of some p-op, notification is made of normal end.

The ⌈control bit renewal⌋ is used with all DEC rest p-ops. They are p-ops directly or indirectly affecting all or any of IF, D and B bits (found in Eflags and in various segment descriptors). If the AP determines (one or plural) new values of affected (one or plural) bits, this end is used for sending renewal values to DEC.

Attention should be paid to the fact that this is not a true end, particularly, a p-op causing a control bit change is not completed. A common p-op end does not require a control bit renewal yet and must occur after the control bit renewal. (The control bit renewal itself must follow a precedent p-op). Attention should also be paid to the fact that if the control bit renewal is received, the DEC may continue a p-op issuance any-time independent of the end of the p-op. With this concept, if new values of affected control bits are known in the processing of p-op, the AP immediately sends the renewal to the DEC and continues processing the p-op.

Since the above control bits express programmer visible bits, the AP and DEC must potentially back out changes for these bits. When the AP notifies the control bit renewal to avoid it (without giving a significant impact on properties) (not later), it changes a master copy of these bits and delays these two actions until the p-op becomes the oldest pending p-op. Essentially, the notification of the control bit renewal suggests that the AP rest before giving notice of the renewal.

The second shape of the control bit renewal is similar to the first shape, but the transfer of renewal also indicates both or either of⌈mis-predicted address⌋and D bits. This is used in transfer of control bits for which the DEC predicted a target address (and took D bits to be unchanged). If the (physical) target address predicted by DEC for transferring the control p-op is not right (i. e., different from a (physical) target address generated by AP), the AP must notifies it and sends renewal values of D bits. Of course, the AP must also send an address renewal (i.e., a right target address) to a cache tag.

The AP sends a right target address through the PAdrBus and carries out everything by giving simultaneous notice of the control bit renewal to both or either of the mis-predicted address and D bits. The renewal is similar to the above first shape relating to sending the renewed control bit value. Moreover, the DEC properly changes some internal state to express the mis-prediction and reopens the instruction incorporation and decoder by using the right address and D bits. As described above, the DEC ensures that the DEC receives the renewed control bits before the next effective macro-instruction can be decoded.

Unlike the first control bit renewal, this is not a true end, and the transfer control is specifically completed. If another abnormality (i. e., exception) is detected, timing is given so that the AP sends a right target address to notify a renewal enables to avoid a notification of⌈control bit renewal end⌋. Namely, the AP either sends an address to notify a renewal end or provides notification of an abnormal end (with an invalid address).

In the case of a page cross causing an abnormality, the PAdrBus transfer does not occur. This can be due to either or both of segment overline (causing common protection trouble) and page trouble. The AP provides notification of an abnormal end and indicates that trouble has occurred. If it is truly required that an instruction execution intersects the page boundary, special processing is started. If viewed from the AP, the processing and end of a page cross request is independent of surrounding p-ops. It is of great importance that the DEC properly gives priority to the exception in a page cross for the flow of a p-op and p-op exception.

The abnormal ends for exceptions provide notification of exceptions defined by corresponding architecture. In case of two exceptions (e. g.,⌈common protection⌋ troubles), there exist a pair of end Ids for notifying the exceptions. One is commonly used, and the other is used in some special environments where it is necessary to differentiate abnormal ends caused by functional units (i. e., IEU and NP) so long as they have different priority orders.

Attention should be paid to the fact that some of the abnormal ends are related to specific macro-instructions. Specifically, ⌈387 unavailable⌋, ⌈invalid operation code⌋ and ⌈common protection⌋ (code = 1111 0100) ends are notified on first p-ops of related p-op sequence. The ⌈common protection⌋ end (code = 1111 0100) and ⌈debug⌋ end (code = 1111 0101 for debug trouble) are notified on first p-ops of macro-instruction sequence. The ⌈debug⌋ end (code = 1111 0001 for debug trap) are notified on first p-ops of macro-instruction and state switch sequences.

MCC end bus

An end bus **65** of MCC **25** is a 1-bit bus using a standard CMOS style time-sharing I/O. An actual signal transfer occurs at the $\phi 1$ - $\phi 2$ boundary (i. e., the MCC end is a $\phi 2$ transfer). Transfers at other phase boundaries are not defined. This bus is used for notifying the end of normal memory write directly generated from p-ops. Ends for memory read, system memory reference and other refer-ences (e. g., I/O) are not produced.

The MCC receives memory reference addresses in order (relative to the issuance order of p-ops causing the memory reference) from AP. The MCC must end the memory write in this order. Therefore, an explicit transfer of p-op tags for notifying the ends is unnecessary. The back end of DEC which monitors end buses based on the ordered end of write predicts which p-op tag is combined with the next end from the MCC.

Notification is provided of the end of memory write when the addresses are received from AP and arranged in an appropriate write reservation queue, regardless of when the MCC receives the data and when the write issues a queue. The read, change and the write of write operations by p-op are also ended. In case of causing a wrong alignment or p-ops greater than 4-bit memory write, the AP must produce more than one mis-aligned address. Notification is given of the end of such a p-op write if the final address is arranged into to the reservation queue.

The AP produces a confident end of p-ops causing memory write without sticking to the MCC end of the p-ops. This occurs when the AP transfers the final address of one or more words aligned to the MCC via PAdrBus. Since the MCC can commonly arrange (one or plural) addresses, the MCC is usually unnecessary to indicate reception of the write addresses. However, in such a case that the MCC cannot arrange the addresses in an appropriate write reservation queue (because the queue is full or the queue is overlapped with a precedent write in one queue), an end with the MCC is necessary. In these later cases, the end is delayed to prevent DEC from advancing the p-op issuance.

If the MCC does have a confident end capable of being delayed, the following can occur. If the AP ends a p-op, the DEC firmly believes that the p-op producing the write completes and is safe in a write reservation queue. The DEC goes forward to the issuance of 7 or abnormal p-op tags more than tags corresponding to these write addresses. Here, the MCC has such problems as the processing of truncations, matching of data and addresses, processing of overlapped memory reads as well as execution of writes into caches.

Accordingly, the MCC has a capacity of delaying the arrangement of addresses having an overlap problem in a queue (of course, AP also has a capacity of delaying to send more addresses). The addresses are delayed by MCC while the end of write with MCC is also delayed (then it is the final address of one p-ops write). The MCC provides notification of the end simultaneously by finally arranging the addresses in an appropriate queue.

All predicted functional units except for MCC completely end a p-op whose end from MMC is predicted, meanwhile the DEC continuously considers the p-op to be unfinished. Essentially, the DEC treats the MCC end of the p-op on similar terms as the end of other functional units so long as a time that the p-op can be withdrawn is related.

As long as relating to the notification of a normal end of an MCC alone, a direct dialog with abnormal ends provided by other functional units (AP, IEU, NP) does not exist. Even a p-op whose end from the MMC is predicted cannot necessarily be completed indirectly by the MCC. When the AP abnormally ends a p-op and does not (probably cannot) produce all of the addresses of correlated memory write, the DEC behaves properly. Namely, the DEC reorganizes these cases, does not delay processing abnormal ends of an MCC, and then continues to properly track the p-op tags of pending memory write.

There is also a special state in which the AP normally ends a p-op but does not produce a corresponding memory write. In these cases, the AP indicates to DEC that⌈normal end but no write⌋ is notified and the write is not issued.

NP end bus

In summary, the NP end is a 2-bit bus (on the premise that p-ops are completed in order) and provides notification of an exception of floating decimal point match. A logic for containing optionally selected NP in CPU is provided, but a detailed explanation is omitted.

Register re-allocation

As described above, a mechanism used for returning the state of the CPU to a necessary phase for outflowing instructions is the register re-allocation. This technique is inevitably associated by mapping an assembly of physical registers greater than an assembly of programmer visible (i. e., virtual) registers. The number of physical registers exceeds the number of virtual registers by the

maximum of p-ops capable of at least permitting it to be pending and changing the registers. This technique is applied to both general-purpose register files and segment register files.

A specific macro-instruction architecture (80386) provides 8 virtual general-purpose registers under the name of VR0 – VR7 and 6 segment registers. As described above, at most 7 p-ops in total and at most two p-ops with changed segment registers are allowed to be pending. In harmony with it, the AP **15** contains an assembly of 15 physical general-purpose registers under the name of PR1 – PR15 (wrong names FR 1 – FR15 in original document, translator) and 8 physical segment registers while an IEU **17** contains 15 physical general-purpose registers. The physical register PR0 exists in IEU but is used for other purposes.

Fig. 5 is a schematic chart for mapping from the virtual registers VR0 – VR7 and physical registers PR1 – PR15. The physical registers have corresponding effective bits schematically shown by "V". These effective bits are used by functional units as follows. To support the general-purpose register re-allocation, a back end register re-allocation logic **175** maintains a pointer assembly array **177** and the free list array **178**. The pointer assembly array and the free list array provide 8 lists of storage, respectively, and each of these lists has a 3-bit index corresponding to the least significant 3 bits of unfinished p-op tags. Each pointer assembly and each the free list are expressed by a column in the chart, respectively.

The pointer assembly and the free list for given indices maintain a state immediately before the issuance of p-ops having tags corresponding to the indicies. The pointer assembly and the free list contain 8 entries corresponding to the virtual registers VR0 – VR7, each entry contains a pointer designating one of physical registers. The free list contains 7 entries including pointers which designates physical register not designated by one member of the pointer assembly.

Consider an initial state before the issuance of a p-op with a tag = 0. In this initial state, VR0 is mapped in PR8, VR1 in PR7, VR2 in PR6,], then VR7 in PR1. The free list contains 7 pointers designating the PR9 – PR 15, the PR is the head of list and the PR15 is the tail of list. This state is stored in a column of entries with a tag = 0.

Consider the following typical series of three p-ops with tags = 0, 1, and 2.
tag = 0: VR0 = VR0 + VR3
tag = 1: VR3 = VR3 + VR5
tag = 2: VR4 = VR0 + VR3

Since the VR0 is mapped on the PR8, a p-op(0) cannot change the PR8 until it is established that the p-op(0) becomes possible to complete. Accordingly, mapping existing before the start of p-op(0) must be changed so that the VR0 is mapped on a physical register in the free list. Since the PR9 is the head of the free list, the VR0 is mapped onto the PR9. Since the PR8 does not stand at the head of the free list until 8 p-ops are issued and the p-op(0) is withdrawn, it is arranged at the tail of the free list. Other items in the free list go forward to the head. Accordingly, an actual p-op issued with the tag = 0 is PR9 = PR8 + PR5.

The next p-op, i. e., p-op(1) changes the VR3. The VR3 is mapped onto a physical register at the head of the free list, i. e., PR10 to make the back-out of this p-op possible. The PR5 is arranged at the tail of the free list, and the PR11 goes forward to the head of the free list. An actual p-op issued with the tag = 1 is PR10 = PR5 + PR3.

The p-op(2) changes the VR4. Accordingly, the VR4 is mapped in the physical register PR11, and the VR5 is arranged at the tail of the free list. The PR5 is arranged at the tail of the free list. An actual p-op issued with the tag = 2 is PR11 = PR9 + PR10.

If a p-op changing a physical register arrives at some functional unit, effective bits of the register are cleared (invalidated) and set up (validated) only when the p-op is completed. This is necessary to ensure that there exist right data for a late p-op that seems to read a physical register. In the described specific examples, the p-op(0) changes PR9, and the p-op(0) changes PR10. Since the p-op(2) requires contents of the PR9 and PR10, it must have effective source registers (PR9 and PR10) before it becomes executable. It occurs only when the p-op(1) and p-op(2) are completed. Attention should be paid to the fact that if any p-ops are swept away, the p-op(0) and p-op(1) are not withdrawn because the p-op(2) is also swept.

Write queue in data cache subsystem

Fig. 6 is a block diagram of MCC **25** providing the control of a data cache subsystem. This job connects write addresses generated by AP **15** and delivered via the PAdrBus with corresponding data generated by any of several chips and delivered via the DXBux **58**, byte aligns write data (right-justified in a 32-bit double-length word) and byte addresses assigned by AP, inspects the memory data dependency on same addresses between write and subsequent read and, if data become available, short-circuits them immediately, maintains the compactness of execution (coherence) by a write operation until ensuring that p-ops generating the write action ends in success and, if necessary, does not change main memory and cache itself to make it possible to truncate the write operation.

The data cache subsystem handles data operations of three categories. A normal data access is a data access assigned by a programmer except for one performed by NP (if any). The other two categories are system access and NP access. Data in the categories read from memories must express a write made by early p-ops of any category, but writes of different categories can be processed non-synchronously. Namely, near writes (to execution order) of different categories do not change same address or, if they are carried out, a non-synchronous effect of writes between the categories is mild.

The MCC **25** contains plural queue structure including a write reservation queue (WRESQ) **300** combined with a write buffer **302** and a multiplexer **303**, a system write queue (SYSWQ) **305** combined with a system buffer **307**, an NP buffer **307**, and a write reservation queue (NPWQ) **310**. The WRESQ **300** plays a role only in a normal data access. It performs all aforesaid functions comprising that each write data (may be a single byte, 16-bit word or 32-bit double-length word, but always right-justified in a single 32-bit double length word from an execution unit to arrive) is aligned so that it is indicated by (one or plural) corresponding addresses capable of assigning the alignment in a memory on any byte boundary, a memory data dependency on same address between the write of any category and subsequent read is inspected.

The SYSWQ **305** buffers system writes until p-ops generating the system writes end in success and they are written into a memory. It provides at most 4 unfinished system writes. System accesses are accesses performed by AP to gain accesses on a hidden system structure (page dictionary entries, page table entries, segment descriptors and task state segment data). All system writes occur as single double-length word read ≅ change ≅ write operations for setting ⌈gained access⌋ or ⌈in dialog⌋. Since the AP does not perform out of order execution, all system accesses generate in order. More-over, the system writes generate from the read ≅ change ≅ write operations, therefore addresses must arrive at MCC in front of write data.

The NPWQ **310** buffers 8 NP write addresses (sufficient for holding the results of two NP p-ops). NP data accesses instructed from NP are different from normal data accesses in three main points. Namely, a single NP p-op can perform both or either of reading/writing data till 10 bytes, but a normal p-op can gain an access to at most 4 byte data. Accordingly, the NP can perform a multiple-length word transfer to perform a write operation assigned by a single p-op. Data for the NP p-op always arrive at MCC in order (i. e., by same sequence as addresses arrive).

The WRESQ **300** is the most complicated write queue and performs the p-op end and truncation processing described later. The WRESQ comprises complicated data receiving 8 entries and an instruction buffer. Each entry contains 4-byte wide data registers which combines a 30-bit wide content-addressable memory (CAM) register for double-length word addresses (a double-length word is 32-bit data), a numerical comparison logic and a 4-bit exclusive tag CAM containing⌈final bits⌋and⌈released bits⌋, and a control logic contains effective bits for data bytes and⌈existent bits⌋ for all data registers.

The WRESQ receives memory addresses for data access from a FIFO buffer called data physical address queue (PAdrQ) (These addresses are buffered by PAdrQ if they arrive from AP). Each address is accompanied by a p-op tag generating an address, a 4-bit byte-capable mask indicating bytes of double-length words transferred to and/or therefrom and⌈final bits⌋indicating whether the addresses are final ones generated by the p-ops.

Each address received from PAdrQ for write or read ≡ change ≡ write access is associatively com-pared with all addresses already input into WRESQ having an effective bit assembly in any byte position indicated by the byte-capable bits accompanied with the address. If any indicating a write which already overlaps in WRESQ is found, a processing of new address into WRESQ must be contrasted until the overlapping write is written into a memory and removed from WRESQ.

In this case, MCC must be interrupted until the position is written into the memory and receives many addresses by the write queue. This is called pipeline function stop, the MCC enables the address to return into PAdrQ in this case, if this structure shows a warning sign of overflow, the MCC locks the PAdrBus and obstructs to issue more addresses than AP. If this is not the case, the pipeline function stop is not requested, or such a function stop is solved by removing overlapping entries and then the new addresses are allocated to positions in WRESQ.

The positions in the WRESQ **310** are selected for the allocation of a round robin mode with an allocation counter. If the selected position is free, the addresses are copied into an ⌈address CAM⌋, the tags and ⌈final bits⌋ are copied into a Tag CAM, 4 ⌈existent bits⌋ and ⌈released⌋ bits are set to 0, and 4 ⌈effective⌋ bits are set in response to the byte-capable bits assigning bytes of written double-length words. On the other hand, if the WRESQ position is still not in use when picking it up for re-allocation (one or more effective bits are indicated by setting them in that position), the MCC must interrupt the receipt of more addresses (functionally stops pipelines) until this position is written into a memory.

In a clock period of writing new entries into WRESQ or thereafter, data are written into a data byte in which the effective bits are set up. An execution unit does not guarantee that the AP transmits addresses before written data are provided, and also not guarantee that the MCC itself can process the addresses in order they arrive. Accordingly, the data probably have sent to MCC already before a WRESQ is established. An 8-entry WBuf **302** receives it. This WBuf is arranged between DXBus (a bus for delivering write data to MCC) and input of WRESQ itself. The data arriving at DXBus are identified by type of operation (normal memory write if addressed in WRSEQ) communicating them and p-op tags that they generate.

If the normal memory write data arrive at DXBus, they are stored in 32-bit WBuf addressed by the least significant 3 bits of the 4-bit p-op tag. The most significant bits of this p-op tag is stored with an entry, and ⌈existent bits⌋ are set up for the entry (unless Tag CAM bits as described later gene-rate). Simultaneously, the tag is searched in the Tag CAM of WRSEQ. If a position (or two adjacent positions) for the data are found in WRSEQ containing one position set up by ⌈final bits⌋, the data are immediately written in the position (or these positions) (the ⌈existent bits⌋ of WBuf entry have not set up yet in this case). Similarly, if an address set up by the ⌈final bits⌋ is input into WRSEQ, a WRSEQ entry corresponding to the p-op tag generated by the address is questioned, if the ⌈existent⌋ bits have setup, all the data are copied into the WBuf entry, the WRSEQ ⌈existent⌋ bits are cleared.

According to the aforesaid two mechanisms, when both data and address exist independently of which of data or address arrives first or even if they arrive at the same time, both the data and address are input into WRSEQ, the ⌈existent⌋ bits of WBuf entry for the p-op are cleared, and (one or plural) ⌈existent⌋ bits of (one or plural) WBuf entries are set up. At this time, the WBuf position becomes free for reuse. Since the data can arrive out of order to the address, a data register and two indepen-dent paths to the ⌈existent⌋ bits of WBuf are provided so that a processing can generate as early as possible. One issued from WBuf can be written in the position, and the address corresponding to this position (selected by a round robin counter) is written simultaneously. The other directly from a DXBus interface can be written in (one or plural) positions identified by Tag CAM. Thereby, they can be written in a new entry in a clock cycle same as a clock cycle in which newly arriving data from DXBus in the new entry formerly established relative to newly arriving address and data from WBuf.

The data input into WRESQ passes through a rotator that is byte aligned with the data in a byte position same as a byte position probable occupied in memory by the data. Another rotator is pro-vided for each of the two data paths into WRESQ. The number of adjacent ⌈effective⌋ bits having a value of 0 by counting it from the least significant byte position of (first) WRESQ entry (of mostly two adjacent entries) represents a number of byte position to the left where the data for alignment must be rotated before the data write into WRESQ occurs. If the precedent WRESQ position also contains an address for same p-op flag, a logic combined with the ⌈effective⌋ bits is not limited to the case, and this data is fed to a barrel shifter by gating ⌈effective⌋ bits of some position.

If the data is written in some position in WRESQ, it is written in any adjacent positions having same tag value (if addressed by Tag CAM) or also in a position adjoining to direction of previous entry allocation and invalidated by the ⌈final⌋ bits (if allocated with a new entry and addressed by a counter). Since the written data is at most 4-byte wide, a sector is rotated on a byte scale to match the data with byte position of one double-length word and then two double-length words are written, and all 4 bytes are simultaneously written in appropriate positions for a write operation which is not matched by bestriding the boundary of double-length words in a memory.

Normal category addresses set up by the⌈final⌋ bits is abstracted from PAdrQ, the MCC feeds an MCC end signal to DEC. These addresses are processed in order (i. e., in order same as p-ops issued from DEC and producing the addresses), the DEC knows which p-ops produce normal memory addresses, and the DEC can clearly correspond the MCC end to the p-op even if the MCC end does not explicitly contains (one or plural) addressed p-op tags. The DEC can guarantee by the end from MCC that more than 8 Wbufs are necessary for receiving data from all the p-ops in which the WRESQ entries have not established yet in the worst case and that independent data and addresses can be properly outflowed in case of truncation. Since DEC does not issue more than 7 p-ops except for the oldest p-op which generates a normal address but still not be completed by MCC, this guarantee is obtained.

The address is extracted from PAdrQ and associatively compared (explained in connection with write addresses) with all addresses formerly input into WRESQ (and also into other two write queues). As described above, the overlapping of arrival write addresses and existent WRESQ entries causes the pipeline functional stop until early entries are written into a memory and removed from the write queues. However, even if a different part of same double-length words is changed, non-overlapping writes can be input into the queues. Addresses for read operation and read ≅ change ≅ write operations (address read) are also associatively compared with write queue entries. This comparison is performed for each byte which is determined by a logical product of byte-capable bits of read address and corresponding bits of queue entry.

If there is no WRESQ entry addressing a byte assigned by a read address or if⌈existent⌋ bits of each entry (write ≅ queue hit) addressing a byte assigned by a read address are set up, the MCC notifies it to a DCI 37 to perform a normal cache search of the address. (Any cache access brings about a necessity of a delay in case of cache miss or main memory operation for searching requested data).

On the other hand, if a read address is hit in one or more write ≅ queues in which⌈existent⌋ bits have not set up yet, the processing of address from PAdrQ must be interrupted (pipeline functional stop) until data for all these entries are received. If the functional stop is solved and cache data are available,

the MCC gates only bits, which do not have ⌈effective⌋ bits instructed in DCI and set up by the write ≅ queue hit, on the DIOBus 57. Other bits selected by the ⌈effective⌋ bits of all address-hit write ≅ queue entries are driven into the write ≅ queues and onto DIOBus by MCC. Accordingly, write data which have not been sent to memory can be short-circuited to subsequent reads. Since the second write for a byte in which the write becomes unfinished in the write queue is received and the pipelines are functionally stopped, more than one entries addressing provided bytes of data cannot exist, but several entries feeding different bytes of same double-length word can exist. The write ≅ queue combines the ⌈effective⌋ bits from all these entries, selects data and drives them onto DIOBus.

Similar to other units of CPU, the MCC must track the state of tags fed from DEC via the tag state bus. The DEC acquires advices of a p-op tag (OOTag) or a truncation tag (ATag) and transmits one of two message type onto the tag state bus in each clock cycle. WRESQ maintains a pointer indicating the oldest entry called ⌈oldest entry pointer⌋ (OEP) . Some entry is unqualified as it is to a write into memory until it becomes older than OOTag. In each cycle receiving the OOTag, the OOTag is compared with tag CAM contents of each write queue entry in which one or more ⌈effective⌋ bits are set up and ⌈effective⌋ bits are not set up. Tag comparison is performed by subtracting the OOTag from 4-bit tag of the entry by use of 4-bit two complementary operations. Since tags are issued by binary counting sequences (0000, 0001, ⌋, 1110, 1111, 0000, ⌋) and no more than 7 tags are pending at a time, the value of OOTag can jump to at most (if the seven pending p-ops withdraw and a new p-op is issued in the same cycle) 8 from one cycle to the next cycle. Accordingly, if the value of most significant bits of a difference obtained by subtracting the OOTag from the tag of entry is "1", it shows that the tag of entry is 1 to p-ops older than OOTag because it is not 8 or more younger than OOTag. In this manner, ⌈released⌋ bits of the entry are set up for each entry found to be younger than OOTag. When the ⌈released⌋ bits of entry indicated by OEP are set up, their ⌈existent⌋ bits are set up and then one or more ⌈effective⌋ bits are set up, then the entry can be written into both or one of cache and main memory only in the case. If a write occurs, the ⌈effective⌋ bits of entry are cleared, and OEP can go forward to the next serial entry in which one or more ⌈effective⌋ bits (if any) are set up.

If the DEC provides notification of truncation, the ATag inspect on p-op tag fields in all queues containing PAdrQ, WRESQ and other two write queues. This inspection is same as the inspection determining a time that an entry can be released, namely, it is performed by subtracting ATag from a tag field assigned in the queues. If the tag field of queue entry is greater (older) than ATag, the entry is maintained in the queue, if not so, its (one or plural)⌈effective⌋ bits are cleared. A pointer probably must also be adjusted depending upon embodiment of a control logic of queue. In case of WRESQ, if an entry is deleted, moved back to an entry in which an allocation pointer is deleted earliest and then moved through OEP, the OEP moves to an entry ahead of the allocation pointer.

Although same inspection is also carried out for other similar structure which is addressed by the entry of WBuf and tag value combined with WRESQ, since the address of entry in WBuf is simply the inferior bits of the tag and only the most significant bits (MSB) of tag of the entry are stored in the entry itself, it is sufficient only by resetting⌈effective⌋ bits of all entries having 3 bits greater than or equal to inferior 3 bits of ATag and stored MSB equal to MSB of ATag or having addresses smaller than inferior 3 bits of ATag and MSB relative to MSB of ATag.

Similar to all functional units of CPU, the MCC neglects data presented on an internal bus in the truncation cycle and, if they are still not appropriate, retransmits the sent data. Accordingly, in some single cycle, the MCC (and remains of CPU) reset the data with confidence to a state having a p-op which supports a tag greater than or equal to ATag still having no issuance.

Processing of pseudo Op in IEU

Fig. 7 is a block diagram of IEU **17**. The IEU realizes two data paths, i. e., a single cycle data path **400** and a multiple cycle data path **405**. The single cycle data path executes all interger instructions capable of completing one cycle such as addition, subtraction and shift, etc. The multiple cycle data path executes all integer p-ops which needs plural cycles such as multiplication, division and ASCII and decimal calculation mechanism. The two data paths uses a common register **410** comprising physical register mapped with virtual registers as explained on register re-allocation.

Each data path contains elements combined with a common bus set **412**, and a bus coupler **415** separates a space between the two data paths. The single cycle data path comprises a general-purpose ALU **420**, a barrel shifter **422** and a special logic **425** for sign propagation, precedent 0 and 1 directions, etc. The multiple cycle data path comprises a multiplication-division circuit **430** (8 x 32 multiplier array) and a circuit **435** for ASCII and decimal number adjustment.

Input p-ops are received from a p-op bus **52** and guided to a p-op queue **450**. A multiplexer **452** selects a p-op executed in the queue, and the executed p-op is communicated to a single cycle con-trol logic **455** (realized by PLA). In case of single cycle p-op, the control logic **455** controls the elements of single cycle data path. In case of multiple cycle data path p-op, the control logic **455** controls the elements of multiple cycle data path and feed addresses to a macrocode ROM **460**. The macrocode ROM **460** provides a control of subsequent cycle of p-op with a multiple cycle control logic **462** (PLA).

In case of ALU p-ops, the results are stored in the registers, an end is input into an end queue **470**, and contents of end queue **470** are delivered onto the IEU end bus. In case of memory write, the results is directly advanced to the DXBus (the end is input into the end queue in this case) or arranged in a DXBus output queue **475** for subsequent output. If the bus is available, the end is input into the end queue.

The depth of p-op **450** is 8. The p-op queue has plural read port and one write port. A queue control logic **480** controls the queue and functions like FIFO (first-in first-out) but also supports an out of order read. The queue control logic indicates whether the queue has an entry. The queue control logic also identifies the position of p-op in queue.

If the p-op queue receives a p-op at the time of empty queue, the p-op is directly decoded to pro-duce an appropriate signal. An execution preparatory inspection is carried out when the decoding of p-op is in progress. This inspection includes some special execution references such as subordination and sequential executions and functional unit serialization. If the p-op fails in the execution preparatory

inspection, some or all control signals become disabled. If the p-op is not executed, the p-op is arranged in the queue.

If an entry exists in the queue, the queue functions like FIFO. The first p-op of the queue and the next young p-op are read out. An execution preparatory logic **482** carries out an inspection on the two p-ops. An execution preparatory inspection on the first p-op of queue includes a data operand subordination. If the first p-op of queue is qualified for the execution preparatory inspection, the p-op is decoded and executed. If the p-op cannot be executed, it is issued for inspection in the next action cycle.

An execution preparatory inspection for the next young p-op in the queue includes a data operand and a flag subordination, an interlock for the first p-op of queue and whether the p-op is main body of a special execution reference (like sequential execution). For example, whether⌈effective⌋ bits are set in a source register requested by the p-op or not is inspected. If the first p-op of queue fails in execution and if the next young p-op in the queue is qualified for the all of execution preparatory inspection, this p-op is decoded and executed. If both the first p-op of queue and the next young p-op in the queue can be executed in success, the first of queue is executed.

The plural read pointers and one write pointer continue to track actions of the queue. If the next young p-op is executed, the corresponding real pointers are renewed so that it points to the next entry in the queue. If the first p-op of queue is executed, the real pointers are renewed so that the first read pointer gets the value of second read pointer and the second read pointer points to the next entry in the queue. The write pointer is used to point to the first empty position. All the pointers are compared with truncation tags in the truncation cycle, and they are set to appropriate values based on the result.

The queue control logic **480** has state bits for each entry in the queue. The state bits are set to "effective" while a new p-op is loaded in the queue. If entries in the p-op queue is swept away in the truncation cycle, appropriate state bits are set to "invalid". The p-op identified for execution is decoded. If the p-op identified for execution is a single cycle p-op, a control signal for the single cycle

data path **400** (register files, ALU, barrel shifter and special logics) is produced by the control logic **455**. The single cycle p-op is executed in a single clock cycle. The multiple cycle data path **405** also does not perform any time-related functions.

If the p-op identified for execution is a multiple cycle p-op, a first state control signal is produced by a single cycle control logic. The single cycle control logic also activates the macrocode ROM **460**. A control signal for residual state is produced from the macrocode ROM and the multiple cycle control logic **462**. The multiple cycle data path **405** performs operations in this time. Multiple cycle actions use the register files from the single cycle data path.

It is possible to perform simultaneous (parallel) execution of p-ops. If the p-op identified for execution is a multiple cycle p-op, considerable performance advantages are obtained by executing the next single cycle p-op from the queue. Single cycle p-ops can be executed by using a single cycle data path and multiple cycle p-ops can be executed by using a multiple cycle data path. If data on multiple cycles or a state flag subordination exists, the single cycle p-op is not executed. In a time that source opposition exists between a multiple cycle p-op and a single cycle p-op (in a write into register files or in state flag renewal), a single cycle p-op is not executed.

A multiple cycle control logic has a state mechanism for identifying the state of actions. The integer execution unit can take four states, i. e., single cycle, multiple cycle, same time or one of play. Buses between the single cycle data path and the multiple cycle data path are cut off by the bus coupler **415** in simultaneous actions. These buses are usually connected in multiple cycle actions. They make it possible to use both or either of data transfer from data files and results from p-op (in front of it for the next p-op).

If some p-op is identified to be executable, the p-op is presented to both or either of the single cycle control logic and the multiple cycle control logic. If a functional unit is found in dialog, the p-op is not executed. It is notified back to the p-op queue to make preparation for executing the logics. An appropriate adjustment is provided to the multiple cycle read pointer.

Usually, the p-op queue, queue control logic and execution preparatory logic are tried to maintain the issuance of p-op based on data operand interlock and special execution reference. Control logics of the functional units in IEU (ALU, barrel shifter, special logics, multiplication-division circuit) solve the source opposition of hardware and perform any operations of single cycle, multiple cycle and same time. If the source opposition is notified by a signal QNEXT and the p-op cannot be executed, a re-issuance is requested by the p-op queue control logic. Flags are tracked by using a flag tag 485.

Conclusion

The preferable embodiment of the present invention is completely described above, but various changes, substitutions and equivalents may also be used. For example, the aforesaid embodiment has been realized by using separated chips for each functional unit, but the basic architecture using distributed pipeline control is effective and useful likewise in a single chip embodiment. Similarly, this specific embodiment executes a specific instruction, it is so designed that other embodiments can execute other instruction assemblies.

Moroever, the specific mechanism for communicating tags to functional units (tag state bus with OOTag or ATag using coded tags) was described, but there exist other possibilities. One possibility in a system capable of making at most n p-ops pending at a time expresses tags as a single set bits in a N-bit vector (however, N is greater than or equal to n). These tags are issued in order so that the collection of pending p-ops are expressed as adjacent groups (in a circulation sense) of single set bits in the N-bit vector. This vector is communicated with the functional units to indicate state. On the other hand, a truncation is notified by a similar type of vector.

Accordingly, the above description and attached drawings do not restrict the scope of the present invention limited by the claims.

Table 1 Pseudo Op Bus Format

In First ϕ1		In First ϕ2	
Bits (single/plural)	Field	Bits (single/plural)	Field
<51..48>	SegReg	<51..48>	DestSegReg
<47..45>		<47>	LastPop
<44..41>	SrcAReg	<46>	(reserved)
<40..37>	IndexReg	<45>	Lock
<36..33>	EASpec	<44..40>	StatMod
<32>	ASize	<39..32>	Imms
<31>	TwoCyc	<31..16>	ImmDispHi
<30..29>	MemRef	<15..0>	ImmDispLo
<28..25>	SrcBReg		
<24..21>	DestReg		
<20>	RegStore		
<19..17>	OperSize		
<16..14>	OperSpec		
<13..4>	Opcode		
<3..0>	PopTag		

In Second ϕ1		In Second ϕ2	
Bits (single/plural)	Field	Bits (single/plural)	Field
<47..14>	(undefined)	<47..32>	(undefined)
<13..4>	Opcode	<31.16>	ImmHi
<3..0>	(undefined)	<15..0>	ImmLo

Table 2 Physical Address Bus Format

In First ϕ1		In First ϕ2	
Bits (single/plural)	Field	Bits (single/plural)	Field
<25>	DTAGReg	<25..23>	Stream
<24>	ITAGReg	<22..20>	Operation
<23>	DecReg	<3.. 0>	InstrNum (= p-op Tag except for Stream 0
<22>	MCCHLd		
<21>	ARReq		

In Second ϕ1		In Second ϕ2	
Bits (single/plural)	Field	Bits (single/plural)	Field
<20>	Lok	<19>	Val
<19>	Trm	<18..4>	Physical address <31..17>
<18..4>	Physical address <16..2>		
<3..0>	Byte selection		

Table 3 DIOBus Format

DIOCtl

In ϕ1		In ϕ2	
Bits (single/plural)	Field	Bits (single/plural)	Field
<4>	Final operand	<4>	RdDATA effective
<3..0>	Frame	<3.. 0>	P-op tag

DIOBus

In ϕ1		In ϕ2	
Bits (single/plural)	Field	Bits (single/plural)	Field
<31..0>	WrData <31..0>	<31..0>	RdDATA <31..0>

Table 4 Data Exchange Bus Format

Cycle 1 ϕ1

Bits	Field
<21>	APReq
<20>	NPHLd
<19>	NPReq

Cycle 2 ϕ1

Bits (single/plural)	Field
<21..20	TT (transfer type)
<19..16>	P-op tag
<15..0>	Data <15..0>

Cycle 2 ϕ2

Bits (single/plural)	Field
<18..16>	MemOp
<15..0>	Data <31..16>

Table 5 IEU End Bus Format

In ϕ2

Bits (single/plural)	Field
<4..2>	Pseudo op tag
<1.. 0>	End Id

A pseudo op tag contains three least significant bits of p-op flag of completed p-op

End Id

Value	Meaning
00	No end
01	Normal end
10	Mis-prediction branch direction
11	end Abnormal end

Table 6 AP End Bus Format

In $\phi 2$

Bits (single/plural)	Field
<3>	End Id, bit <7>
<2>	Id<6>
<1>	Control bit B/Id<5>
<0>	Control bits D/Id<4>

In $\phi 2$ (normal)

Bits (single/plural)	Field
<3>	Control bit I/Id<3>
<2>	Control bit N/Id<2>
<1>	Control bit H/Id<1>
<0>	Control bit S/Id<0>

End Id <7..0>

Value	Meaning
00XX XXXX	No end
01BD INHS	Control bit renewal
10BD INHS	Mis-prediction address/ control bit renewal
110X XXXX	Normal end
1110 0001	Debug
1111 0010	929⊥ code
1111 0100	General protection (instruction sensivity)
1111 0101	Debug (break-point)
1111 0110	Invalid operation code
1111 0111	387 unavailable
1111 1000	Double trouble
1111 1001	Interruption
1111 1010	Invalid TSS
1111 1011	Segment absence
1111 1100	Stack trouble
1111 1101	General protection (excluding instruction)
1111 1110	Page trouble

Table 7 Examples of Sequence of Pseudo Op End and Tracking

	Tag	Pseudo Op	AP Item Tag value	IEU Item Tag value
	3	CHK } AG		
	4	XFE |		
A	5	XFE | AG		
B	6	DEC } AG	3 OK	
	7	XFE |		3 OK
	8	XFE | AG	4 OK	
C	9			6 OK
			5 OK	4 OK
				5 OK
			6 OK	9 OK
			7 OK	
			8 page trouble	
D				7 OK

Fig. 1

3	state			
4	ILEN			
5	control			
6(12)	DIO control			
8	control address			
12 – 15	addresses			
22 (32 +control)	control			control
25	MCC	↔	MCC	↔
32	DIO data			
50	data incorporation			
52 (104)	POP bus			
55	PADR bus			
58	DXBus			
60	AP end bus			
62	IEU end bus			
63	NP end bus			
65	MCC end bus			
j(2)	NP end			
lower left corner	control			

Fig. 2

2	V bits
4	instruction length to AP
5	I register
48	↑ pseudo OP
50	instruction bus
55	PADR bus
100	front end
	truncation information
	end information
102	decoder
	existent instruction byte
	effective instruction byte bits
	instruction length
	truncation information
	vectorization
	holding states
	register re-allocation
	register allocation
	P-OP information
105	back end ↑ P-OP state ↑ interruption
	AP end
	IEU end
	NP end
	MCC end

Fig. 3A

50	instruction bus			
110	rotation/shift			
112	PCs	↓ instruction length	← existent PC	↓ target address
115	control	↑ to decoder ← V bits	↓ end information	→ instruction request
117	stream stack			
120	PADR bus monitor			

Fig. 3B

130	IREG		
132 (from left)	operation code decoding PLAs		operation code decoding PLAs
135	P-OP type decoding	↓ to F.E.	↓ to B.E.
137	instruction length decoding	→ ILEN bus	
140	P-OP assembly logic	→ tag from B. E. ← existent register allocation from	→ register re-allocation to B. E.
142	P-OP output queue	↑ P-OP bus driver	↑ P-OP bus 4/II
145	issuance holding		
147	sequencer		
150	decoding holding	↑ from F.E.	
MPX	↑ vectorization and interruption logic		↑ from B.E.

Fig. 3C

160 (left) P-OP information issuance

(right) P-OP information complete end → state→ oldest tag to FE

(inside) young old

165 holding state calculation → holding states to decoder

170 (left) DPC address stream ID address from F.E. → stream ID history

← stream ID for branch truncation

(right) memory priority order logic response selection → truncation flag

truncation information to decoder

(bottom) selector

(inside) young old

177 PSA 178 FLA

register

segment register re-allocation

re-allocation logic

↑ register re-allocation from decoder ↓ existent allocation

179 tag production

→ decoder ↑ issuance from DEC ↑ truncation flag ↓ flag/state pad

180 end bus logic → to F.E.

(left) → end bus pad → decoding/BADI

↓ interruption logic ↓ interruption pad

(left) ← instruction information ← interruption

Fig. 4A

Fig. 4B

Fig. 4C

Fig. 4D

160 truncation tag = 7

Fig. 5

(left, from top) (right)

tag → physical register

tail

the free list

head

virtual register

Fig. 6

300 WRESQ (8 entries)

302 write buffer (8 entries)

305 SYSWQ (4 entries)

307 system buffer (1 entry)

310 NPWQ (8 entries)

312 NP buffer (4 entries)

320 PADRQ (4 entries)

MPADRST (upper left) 1 FSM (1 entry) pipeline multiplexer

MPIP (left middle) pipeline control ← DIO control ← MCC end

MDXBUS (upper right) DXBUS interface

MRDBUF (lower) read buffer (3 entries)

MMIOST (lower) MMIO register (4 entries)

write bus **20**

short-circuited bus

state renewal bus

(bottom) system bus interruption DIOBUS interface

Fig. 7A

450	P-OP queue ↓ POP bus <47..0>		
455	single cycle PLA'S		
460	multiple cycle P-OP ROM	↓ Complex PLA	
470	end queue		
480	P-OP queue control		
482	Execution preparatory logic		
(most left, from top)	tag state <4..0>	IEU end <4..0>	DIO control <4..0>
	DIO bus <31..0>		
(upper left, from top)	truncation control	end queue control	
(bottom, from left)	DINOUT bus queue	queue control	IMMED operand queue
	queue control	state flag stack ↓ flag register	flag control

Fig. 7B

(top left)	secondary bus	read primary bus	write primary bus
(top right)	bus (415)	coupler	LER
(upper, from left)	register file	ALU	barrel shifter
	special logic	multiplication/division	ASCII decimal adjustment
(middle, from left)	DXBUS input queue	DXBUS queue control	DXBUS output queue
		DXBUS mediation	

[Claims]

(corrected, see Procedural Corrections at the end of document, issued on February 10,)

Claim 1.

A computer processor, characterized by providing:
a unit for issuing a series of operations achieving one unfinished operation, respectively if issued,
multiple functional units capable of executing at least some of the unfinished operations, respectively,
a unit for allocating tags comprising one member of an ordered tag assembly to each unfinished operation so that the relative ages of two unfinished operations can be determined by inspecting the allocated tags,
a unit for determining the time of completing the given unfinished operations, and
a unit for limiting the number of unfinished operations to ensure the uniqueness of the unfinished tags.

Claim 2.

The computer processor referred to in Claim 1, in which the above limiting unit allowed to make at most n unfinishable operations at a time, wherein the above tags are issued in order over some range where they are in a range of more than or equal to 2n, and
the relative ages of two unfinished operations can be determined by a comparison of these tags.

Claim 3.

The computer processor referred to in Claim 1, which also provides
a unit for determining the oldest unfinished operations and
a unit for providing notice of a successful withdrawal of operations to at least some of the above

functional units by preparing tags that mark the boundary between the unfinished operations and the withdrawal operations.

Claim 4.

The computer processor referred to in Claim 3, which also provides
a unit for grouping adjacent operations so that the withdrawal of any operations in a group is performed only when the withdrawal of all operations in that group becomes possible.

Claim 5.

The computer processor referred to in Claim 3, which also provides
a unit for buffering intermediate memory writes in a period such that at least their source operations are unfinished but not withdrawn,
a unit for outflowing buffered writes at the time of outflowing the source operations, and
a unit for completing the arrangement of the buffered writes into a cache or memory when the source operations are withdrawn.

Claim 6.

The computer processor referred to in Claim 5, which also provides a unit for returning the buffered write data to subsequent read operations before the write data are outflowed or arranged in the cache or memory.

Claim 7.

The computer processor referred to in Claim 1 wherein the above issuing unit responds to an input instruction with an instruction set architecture containing m programmer visible registers and at least some operational changes of one of these registers, which also provides a unit for limiting the number

of unfinished register change operations to n, and at least (m + n) physical registers, and
a unit for mapping the programmer visible registers into the physical registers.

Claim 8.

The computer processor referred to in Claim 1, which also provides
a unit for ensuring that the physical registers are not re-used until the operation changing the physical registers has been successfully withdrawn and a unit for restoring the mapping from the imaginary to the physical to a precedent state at the time of detecting an abnormal state and therefore restoring the contents of the programmer visible registers.

Claim 9.

A computer processor, characterized by the fact of improving the interlock bypass and properties by providing multiple writing buffer queues allocated for processing specified types of write, respectively,

a unit for selecting the highest priority order for each write queue and
a unit for selecting the highest priority order queue.

Claim 10.

A computer processor, characterized by providing a unit for transforming instructions in an input stream containing the instructions into a series of operations in response to the input stream, multiple functional units capable of executing at least some of these operations, respectively, a unit for communicating the operations with at least some of the functional units (thus the communicated operations are called unfinished operations), a unit for maintaining information on the end of operations made by the functional units for each unfinished operation,
a unit for determining when each operation combined and communicated with each functional unit

ends and communicating this end information to the maintaining unit together with a tag of opera-tion, a unit for determining the oldest unfinished operation, a unit for communicating the indication of the oldest unfinished operation with the functional unit, a unit for permitting the withdrawal of the operation only in the case that end information on at least the oldest unfinished operation shows that the operations of all the functional units are normally completed in response to the end information of the operation, and
a unit for renewing the indication of the oldest unfinished operation to express that an operation thus withdrawn is not the oldest unfinished operation in response to the withdrawal of at least the oldest unfinished operation.

Claim 11.

The computer processor referred to in Claim 10, which also provides:
a unit for communicating a truncation tag assigning a group of operations to be outflowed to the functional unit in response to information that the given unfinished operation has been abnormally completed, and

a unit for outflowing all unfinished operations combined with the functional units and assigned by truncation,
a unit for deleting the assignment of unfinished operations from the operation operations assigned by the truncation, and
a unit for starting the allocation of tags started from a value equal to the tag of the outflowed oldest operation in the tag allocation unit.

Claim 12.

A computer processor, characterized by providing a unit for transforming instructions in an input stream containing instructions relating to a series of operations in response to the input stream, multiple functional units capable of executing at least some of these operations,

a unit for communicating these operations with at least some of the functional units (thus the communicated operations are called unfinished operations), and a unit for limiting the number of unfinished operations to a prescribed maximum,

a unit for allocating tags to each unfinished operation in order,

a unit for maintaining information on the end of operations made by the functional units for each unfinished operation, a unit for determining when each operation combined and communicating with each functional unit ends and which communicates the end information to the maintenance unit together with an operation tag,

a unit for communicating the indication of the unfinished operations with the functional units,

a unit for withdrawing the normally completed operations in order in response to the end information from the functional units,

a unit for providing instructions to the functional unit that at least this given unfinished operations and all later operations are outflowed in response to information that the given unfinished operations have been abnormally completed,

a unit for outflowing all the unfinished operations combined with the functional units and assigned by the instruction unit, and

a unit for starting the allocation of tags started from a value equal to the tag of the outflowed earliest operation in the tag allocation unit.

Claim 13.

A method for controlling pipelining operations in a computer processor containing multiple functional units capable of executing at least some of unfinished the operations, respectively, characterized by being provided with a step for allocating tags comprising one member of ordered tag assembly to each unfinished operation so that the relative ages of two unfinished operations can be determined by inspecting the allocated tags,

a step for determining the time of completing the given unfinished operations, and

a step for limiting the number of unfinished operations to ensure the uniqueness of the unfinished

tags.

Claim 14.

The method referred to in Claim 13 with at least some branch operations in the above operations comprising a step for predicting the results of unfinished branch operations,
a step for detecting mis-predictions about the unfinished branch operations, and
a step for outflowing all unfinished operations issued as the result of mis-predicted branch operations.

Claim 15.

The method referred to in Claim 13 wherein the above issuing unit responds to an input instruction with an instruction set architecture containing m programmer visible registers and at least some operations change one of these registers, which also provides a step for limiting the number of unfinished register change operations to n,
a step for preparing at least (m + n) physical registers, and
a unit for mapping the programmer visible registers into the physical registers.

Claim 16.

The method referred to in Claim 15, which also provides which also provides:
a step for ensuring that the physical registers are not re-used until the operations changing the physical registers have been successfully, and
a step for restoring the mapping from imaginary to the physical to a precedent state at the time of detecting an abnormal state and therefore restoring contents of the programmer visible registers.